Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Borut Ajdič

Človeške aktivnosti v izvedljivih poslovnih procesih na osnovi WS-BPEL Extension for People

diplomsko delo
na univerzitetnem študiju

Mentor: prof. dr. Matjaž Branko Jurič

Ljubljana, 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani **Borut Ajdič**, z vpisno številko **63060005**, sem avtor diplomskega dela z naslovom:

Človeške aktivnosti v izvedljivih poslovnih procesih na osnovi WS-BPEL Extension for People

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom **prof. dr. Matjaža Branka Juriča**,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 03.09.2015 Podpis avtorja:

Zahvala

Zahvaljujem se mentorju, prof. dr. Matjažu B. Juriču, za pomoč in nasvete pri izdelavi diplomske naloge.

Zahvala gre tudi moji družini in Barbari za vso podporo in spodbudo med študijem.

Kazalo

Seznam uporabljenih kratic in simbolov

BPEL Business Process Execution Language (izvršljiv jezik po OASIS standardu)

BPEL4People Business Process Execution Language for People (razširitev jezika BPEL z uporabniško aktivnostjo)

BPM Business Process Management (upravljanje poslovnih procesov)

BPMN Business Process Model and Notation (grafična notacija za modeliranje poslovnih procesov in delovnih tokov)

BPS Business Process Server (strežnik za upravljanje in izvajanje procesov, opisanih z jezikom BPEL)

HT Human Tasks (človeška opravila)

HTM Human Task Manager (samostojna programska komponenta za upravljanje s človeškimi opravili)

HTML Hypertext Markup Language (označevalni jezik za oblikovanje večpredstavnostnih dokumentov, ki omogoča povezave znotraj dokumenta)

HTTP Hypertext Transfer Protocol (protokol za izmenjavo večpredstavnostnih vsebin na spletu)

HTTPS Hypertext Transport Protocol Secure Sockets (protokol, ki omogoča varno spletno povezavo)

IdP Identity Provider (ponudnik, ki dodeljuje, vzdržuje in preverja podatke o identitetah)

IPSec Internet Protocol Security (varnostni protokol za zaščito komunikacije)

IP Internet protocol (protokol omrežnega sloja v protokolnem skladu TCP/IP)

JMS Java Messaging Service (storitev za asinhrono izmenjavo kritičnih poslovnih podatkov in dogodkov)

LDAP Lightweight Directory Access Protocol (internetni protokol za dostop do imenikov na arhitekturi odjemalec-strežnik)

LGPL GNU Lesser General Public License (licenca, prvotno mišljena za licenciranje programskih knjižnic)

OASIS Organization for the Advancement of Structured Information Standards (neprofitna organizacija, ki nadzira in skrbi za razvoj odprtih standardov)

PPP Point to Point Protocol (protokol za prenos podatkov, običajno uporabljen za vzpostavitev neposredne povezave med dvema vozliščema v omrežju)

SMTP Simple Mail Transport Protocol (internetni protokol za prenos elektronske pošte)

SOA Service Oriented Architecture (arhitekturni koncept, ki predlaga razvoj aplikativnih rešitev v obliki samostojnih enot, imenovanih storitve)

SOAP Simple Object Access Protocol (standard za spletne storitve, ki temelji na XML)

SSL Secure Socket Layer (protokol, ki omogoča šifrirano povezavo med strežnikom in odjemalcem)

STS Security Token Service (programska oprema, ki temelji na ponudniku identitete, odgovorna za izdajo varnostnih žetonov)

TCP Transmission Control Protocol (povezavni protokol transportnega sloja v protokolnem skladu TCP/IP)

UDDI Universal Description, Discovery and Integration (javni ali privatni register, namenjen objavi in lociranju metapodatkov o spletnih storitvah)

UDP User Datagram Protocol (nepovezavni protokol transportnega sloja v protokolnem skladu TCP/IP)

WSDL Web Services Description Language (opisni jezik spletnih storitev)

XML Extensible Markup Language (razširljivi označevalni jezik)

XPath XML Path Language (poizvedovalni jezik za dostop do podatkov v dokumentih XML)

XSD XML Schema Definition (jezik za opis sheme dokumentov XML)

Povzetek

Podjetja v poslovnem svetu dajejo identifikaciji in optimizaciji poslovnih procesov velik poudarek. Razlog je zagotovo v iskanju konkurenčne prednosti in hitrem odzivu na spremembe na trgu. Za ta namen se uporabljajo avtomatizirani poslovni procesi. Ključni problem avtomatiziranih procesov je pomanjkanje formalnega opisa človeške interakcije s procesom. V diplomskem delu smo opredelili pojem opravila in opisali njegovo vlogo v avtomatiziranem poslovnem procesu. Identificirali smo scenarije uporabe, ki vključujejo ljudi v avtomatiziran poslovni proces na podlagi opravil, definirali življenjski cikel opravila in glavne koncepte človeške aktivnosti. Predstavili smo formalni jezik BPEL4People za podporo človeški interakciji in specifikacijo WS-HumanTask za abstrakten opis opravil. Z orodjem BPM smo na praktičnem primeru implementirali poslovni proces, ki zajame vse glavne koncepte uporabniškega opravila, in po uvedbi opisali izkušnje iz produkcijske uporabe.

Ključne besede:
izvedljivi poslovni procesi, BPEL, BPEL4People, človeška opravila

Abstract

Companies place significant emphasis on business process identification and optimisation. The main reasons for that can be found in identifying a competitive advantage and providing a rapid response to the changes on the market. For these purposes, business process automation is needed. A significant problem in automated business processes is the absence of formal description of human-process interaction. In this thesis, we defined the concept of task and its role in automated business processes. We described scenarios where users are involved in automated business processes based on human tasks, as well as defined human task lifecycle and the main concepts of human activity. We described the BPEL4People formal language to support user interactions and the WS-HumanTask specification for an abstract description of the human task. We have also modelled a practical business process using BPM tools and captured all concepts of human task as well as presented results from production use.

Key words:
business process, BPEL, BPEL4People, Human tasks

Poglavje 1

Uvod

Poslovnega procesa v splošnem največkrat ni mogoče popolnoma avtomatizirati. Tudi pri dobro modeliranem procesu se lahko ta med izvajanjem ujame v nepredvideno stanje. Nepredvidena stanja so nezaželena in jih lahko odpravi le uporabnik. Uporabnik lahko na primer sprejema odločitve, ki pomembno vplivajo na vrstni red izvajanja aktivnosti, ki jih opisuje poslovni proces. Teh odločitev ni mogoče vedno opisati s poslovnimi pravili. Tipičen poslovni proces zahteva tudi aktivno sodelovanje uporabnika. Ta mora opravljati nadzor nad vmesnimi rezultati in sprejemati odločitve oziroma priskrbeti informacije za pravilno nadaljevanje procesa.

- V poslovnem procesu, kjer potrebujemo akcijo odobritve, mora s svojo odločitvijo posredovati človek, ki akcijo odobri ali zavrne. Na primer vodja kadrovske službe odobri ali zavrne prošnjo za letni dopust.

- Obstajajo tudi poslovni procesi, v katerih je treba zbrati določene informacije, na podlagi katerih lahko naredimo poslovno odločitev. Na primer pregledamo in analiziramo papirnati dokument.

- Obvladovati moramo situacije, ko pride do napak v procesu ali izjem, ki jih ni mogoče odpraviti avtomatično. Na primer kontrola veljavnosti kreditne kartice ni uspela.

Med izvajanjem procesa lahko nastanejo situacije, pri katerih so nujne uporabniške interakcije. V diplomski nalogi smo uporabniške interakcije obravnavali kot seznam opravil, ki jih mora opraviti človek za uspešno izvedbo poslovnega procesa. V nadaljevanju smo opisali strukture, ki omogočajo abstrakten opis in vključitev nove, t. i. *uporabniške aktivnosti* v jeziku BPEL (angl. Business Process Execution Language).

V tretjem in četrtem poglavju smo obravnavali možnosti integracije človeka v izvedljivi proces. Odgovorili smo na vprašanja, kako identificirati ustrezne deležnike procesa in kako implementirati človeške interakcije ter pri tem pustiti dovolj možnosti za naknadne prilagoditve izvedljivih procesov poslovnim potrebam.

V petem poglavju smo preko specifikacije WS-HumanTask podrobneje spoznali pojma *opravila* in *obvestila* ter njuno vlogo v izvedljivih procesih. Definirali smo njuna življenjska cikla in opisali glavne koncepte ter pomanjkljivosti. Za podporo zunanjim aplikacijam oziroma odjemalcem smo spoznali različne metode za administracijo in upravljanje opravil.

Na koncu smo v praktičnem delu prikazali nujnost in uporabno vrednost človeške aktivnosti v izvedljivih poslovnih procesih. Na praktičnem primeru poslovnega procesa smo implementirali različne koncepte človeških aktivnosti, vezane na našo problemsko domeno, in spremljali njegovo izvajanje. Za načrtovanje in izvajanje procesa smo uporabili orodja in strežniško infrastrukturo podjetja WSO2.

Cilji naloge so predstaviti koncepte uporabniških aktivnosti v izvedljivih procesih in opisati formalen jezik za njihov opis na abstraktnem nivoju na osnovi WS-BPEL Extension for People. Poiskati tehnološko rešitev za izvajanje človeških opravil in nakazati smernice za lastno implementacijo tovrstnega sistema. Izdelati in opisati praktičen primer poslovnega procesa ter med delovanjem preveriti njegove poslovne učinke.

Poglavje 2

Arhitektura sistema in tehnologije

2.1 Izvedljivi poslovni procesi

Za boljše razumevanje problemske domene moramo najprej definirati poslovni proces. Poslovni proces je množica logično povezanih opravil, ki so lahko avtomatizirana ali jih izvede določena oseba oziroma skupina oseb in imajo za poslovni sistem določeno poslovno vrednost [13]. Danes skoraj ni podjetja, ki se v praksi ne bi srečalo z informatizacijo poslovnih procesov in njihovo optimizacijo.

Eden ključnih razlogov za to, da podjetja v poslovnem svetu dajejo identifikaciji in optimizaciji poslovnih procesov tako velik poudarek, je zagotovo v iskanju konkurenčne prednosti in hitrem odzivu na spremembe na trgu. V začetku devetdesetih let se je, kot odgovor na potrebo po spreminjanju poslovanja oziroma poslovnih procesov v smeri večjega doseganja konkurenčne prednosti, pojavil nov pristop pri upravljanju sprememb poslovnega procesa BPM (angl. Business Process Management). Ne dolgo za tem se je na trgu pojavil tudi standard za modeliranje tokov poslovnih procesov in delovnih tokov BPMN (angl. Business Process Model and Notation).

Izvedljive poslovne procese umeščamo med zasebne poslovne procese. To

so notranji procesi, ki pripadajo določeni organizaciji in so predstavljeni z diagrami poteka. Izvedljivi procesi so modelirani z namenom poznejše implementacije in imajo vnaprej določen postopek izvajanja ter vhodna in izhodna sporočila [16, 14].

V diplomskem delu smo izvedljive poslovne procese modelirali v jeziku BPEL. Človeške aktivnosti (angl. People activities) v izvedljivih poslovnih procesih pa smo definirali z njegovo razširitvijo na osnovi BPEL4People (angl. WS-BPEL Extension for people).

2.2 Vključevanje ljudi v izvedljive poslovne procese na podlagi opravil

Spletne storitve so postale splošno uveljavljen standard za distribuirane poslovne aplikacije. Distribuirane aplikacije so tiste, ki se izvajajo na različnih fizičnih računalnikih in pri tem uporabljajo podatke, ki so na različnih fizičnih lokacijah. Prinašajo maksimalno povezljivost sistemov in uporabljajo fleksibilno arhitekturo. Prednost spletnih storitev je tudi, da pred klicateljem skrijejo implementacijo in kompleksnost poslovne logike.

Jezik BPEL, s katerim formaliziramo opis izvedljivih poslovnih procesov, omogoča orkestracijo spletnih storitev. To pomeni, da je usmerjen izključno v integracijo oziroma komunikacijo med spletnimi storitvami. V svoji zasnovi ne omogoča aktivnosti, ki bi formalno opisala komunikacijo uporabnika s procesom. Primer take komunikacije je sprejem poslovne odločitve v določeni fazi procesa od referenta.

Če želimo tako interakcijo definirati v okviru jezika BPEL, potrebujemo nekakšno zunanjo aktivnost, ki temelji na arhitekturi spletnih storitev in jo je mogoče klicati znotraj izvedljivega poslovnega procesa. Poleg tega, da mora taka storitev omogočati sprejem vhodnih podatkov in vračati poslovne rezultate, mora hkrati znati obveščati uporabnike in jim omogočiti vmesnik za obdelavo vhodnih podatkov in sprejemanje poslovnih odločitev, ki vplivajo na potek poslovnega procesa. Potrebujemo torej formalni opis, kako se lahko

ljudje vključujejo v izvedljive poslovne procese.

Glede na navedena dejstva je bila potrebna rešitev, ki omogoča integracijo ljudi v izvedljive poslovne procese. Kako pomembno je to, so prepoznala tudi večja podjetja IBM, SAP in Adobe ter pripravila dokument [4], v katerem so obravnavane različne potrebe, scenariji in rešitve integracije človeka v izvedljive poslovne procese na podlagi razširjenega jezika BPEL4People. Razširitev je aprila 2007 odobrila tudi organizacija za razvoj standardov OASIS (angl. Organization for the Advancement of Structured Information Standards) [22].

2.3 BPM

Disciplina upravljanja s poslovnimi procesi (angl. Business Process Management) združuje ljudi, poslovne procese in tehnologijo. S kombinacijo načrtovanja, modeliranja, avtomatizacije, nadzora in uporabe metrik pri izvajanju poslovnih aktivnosti v organizaciji želimo optimizirati poslovne procese in povečati njihovo dodano vrednost [5].

Za organizacijo je ključnega pomena, da dobro razume svoje poslovanje. Stalna revizija poslovnih procesov je zato pomembna aktivnost, ki omogoča organizaciji primerjavo poslovnih učinkov s svojimi strateškimi cilji in načrtovanje efektivnejšega poslovanja v prihodnosti. Tako se lahko hitreje odziva na potrebe trga, dviga kvaliteto storitev in povečuje zadovoljstvo kupcev.

Glavne prednosti in cilji uvedbe upravljanja poslovnih procesov v organizaciji so:

- identifikacija in formalizacija posameznih procesov in njihova umestitev v širši kontekst na način, da so razumljivi vsem deležnikom procesa,
- vzpostavitev lastništva in skrbništva nad procesi ter določitev odgovornosti za njihovo učinkovito izvajanje tako s finančnega kot tudi s strateškega vidika,

- identifikacija odvisnosti med strategijo, procesi, kadri in tehnološkimi sistemi ter določitev prioritet za prenovo ključnih procesov, ki so naravnani na doseganje strateških ciljev in preprečujejo uspešno poslovanje,
- merjenje učinkovitosti procesov in prilagoditev organizacijske sheme, da bi se povečala učinkovitost zaposlenih.

2.4 SOA (angl. Service Oriented Architecture)

Arhitektura v programerskem svetu je sicer prisotna že od samega začetka razvoja programske opreme, čeprav se v strokovni literaturi kot glavna tema na področju razvoja programske opreme pojavlja šele od druge polovice devetdesetih let. Arhitektura opisuje večje komponente sistema in njihovo medsebojno interakcijo na višjem nivoju. Z namenom zmanjševanja kompleksnosti programske kode in večje uporabnosti obstoječe so se sčasoma oblikovali konstrukti, kot so funkcije, razredi in komponente [9].

Razvila se je nova filozofija v načrtovanju informacijskih rešitev, ki je danes poznana pod pojmom objektno-orientirana arhitektura. Prinesla je svojo množico principov, ki se aplicirajo v različnih tipih relacij med objekti. Le-ti predstavljajo manjše zaključene logične rešitve, ki z medsebojno komunikacijo rezultirajo v končno rešitev. Idejno zasnovo za večino principov načrtovanja s storitvami lahko najdemo ravno v objektnem svetu.

Storitev (angl. Service) kot osnovna komponenta arhitekture SOA je samostojna aplikacija, ki opravlja natančno določeno operacijo. Njen vmesnik opisuje sporočila, ki si jih izmenjujemo preko omrežja z uporabo standardnih protokolov. Deluje avtonomno - tudi izven konteksta aplikacije [26].

Glavni principi načrtovanja sistema s storitvami so:

- Implementacija storitve je za uporabnika skrita. Na voljo je le javno dostopna abstraktna predstavitev informacij, potrebnih za uspešno uporabo storitve.

- Storitev naj omogoča ponovno uporabljivost. Logična rešitev, ki jo storitev predstavlja, naj bo centralizirana samo v eni dostopni točki.
- Storitev naj bo avtonomna pri uporabi resursov v svojem izvajalnem okolju. Deljen dostop do resursov z drugimi sistemi zmanjšuje predvidljivost delovanja storitve.
- Storitev med izvajanjem vzdržuje neko stanje in uporablja resurse. Princip načrtovanja storitev brez stanja je v zmanjševanju beleženja lastne zgodovine izvajane logike in odvisnosti od stanja povezanih storitev.
- Odkrivanje storitev je princip, ki pomaga preprečevati podvajanje implementacije in olajša ponovno uporabljivost storitev. S pomočjo metapodatkov je natančno definiran način dostopnosti in namenskosti storitve.
- Kompozicija (angl. Composition) je pomemben koncept arhitekture SOA in omogoča združevanje storitev v določenem vrstnem redu in po določenih poslovnih pravilih v novo, večjo storitev, ki v celoti nudi podporo poslovnemu procesu [13].

Storitveno orientirane arhitekture ponujajo odgovor na potrebo po integraciji poslovnih aplikacij na način, kjer so aktivnosti na modularen način izpostavljene kot storitve in z medsebojno komunikacijo podpirajo poslovni proces. Ključni poudarek storitvenih arhitektur je torej na kvaliteti storitev, zbližanju informacijske tehnologije s kupcem in v reševanju ključnih poslovnih problemov. Prednosti v pravilni implementaciji storitvene arhitekture so v hitrejšem odzivu sistema na potrebe trga in hitrejši povrnitvi stroška investicije na račun večje fleksibilnosti.

2.5 XML

Razširljivi označevalni jezik XML [27] se uporablja za strukturiran opis podatkov in njihovo konsistenčno izmenjavo. Največja prednost zapisa podatkov v jeziku XML je, da ne potrebujemo posebne strukture za shranjevanje podatkov, ampak je struktura definirana in opisana že s samimi podatki. To pomeni, da je dokument XML samopojasnjujoč. Tako omogoča aplikacijsko neodvisnost podatkov in je razumljiv vsem.

Osnovni gradnik XML je **element**, ki je opremljen z začetno in s končno značko (angl. tags). Značke oklepajo vsebino, ki je lahko tudi prazna. Elemente lahko dopolnimo z **atributi**. Atributi opisujejo dodatne informacije o elementih in so sestavljeni iz imena in vrednosti. Dokument, napisan v jeziku XML, mora vsebovati ustrezno deklaracijo in mora pravilno gnezditi elemente v skladu s specifikacijo XML. Le tako ga lahko ustrezno obdeluje program za razčlenjevanje XML. Njegovo pravilno strukturo preverjamo s shemo XML.

Zaradi svoje enostavnosti in neodvisnosti od programske in strojne opreme ter operacijskih sistemov in transportnih protokolov se je tehnologija XML hitro uveljavila v raznih aplikacijah svetovnega spleta.

2.5.1 Shema XML

S pomočjo shem XML preverjamo podatkovno in strukturno veljavnost dokumentov XML. Shema določa, kako so v dokumentu XML lahko uporabljene strukture [31] in podatkovni tipi [32]. Je ločen dokument, v katerem definiramo seznam elementov in atributov, ki se lahko uporabijo v dokumentu XML, ter njihovo medsebojno razmerje, številnost in postavitev.

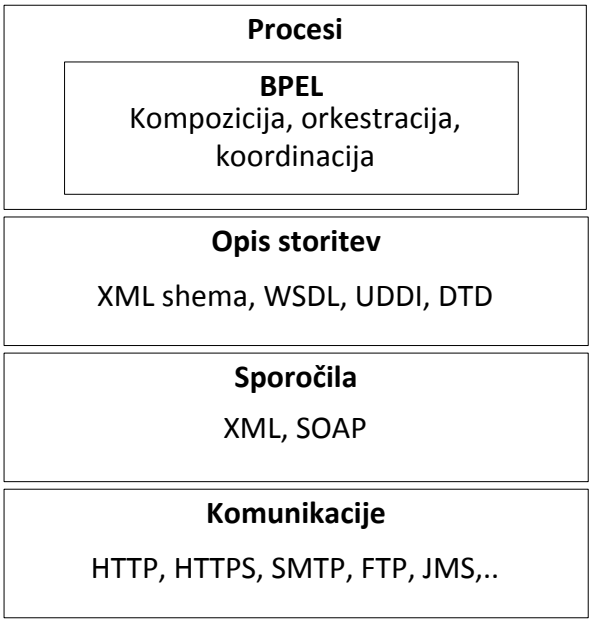
Tudi definicija sheme XML je napisana v jeziku XML in posledično ne potrebuje posebnega vmesnega procesiranja ter je enostavna za učenje. Prednost sheme XML je možnost avtomatičnega kreiranja. Sama po sebi je že dovolj dokumentirana in nad njo enostavno izvajamo transformacije. Je preprosto razširljiva in njene dele lahko uporabimo v drugih shemah. Kreiramo

lahko svoje poljubne podatkovne strukture in jih tudi dedujemo iz obstoječih podatkovnih tipov.

2.6 Spletne storitve

Spletne storitve [28] so programska oprema, ki omogoča izmenjavo podatkov med različnimi sistemi preko omrežja. Na standardiziran način izpostavljajo funkcionalnost in omogočajo dostop do aplikacije kot storitve, s čimer izpolnjujejo eno od zahtev arhitekture SOA. Spletne storitve uporabljajo določen komunikacijski protokol za medsebojno izmenjavo sporočil, ki so definirana s shemami XML. Storitve si predstavljamo kot črne škatle, ki svojo funkcionalnost definirajo in opišejo z vmesniki v jeziku XML.

Povedano drugače: Spletna storitev je sklop funkcionalnosti, ki služijo kot gradniki v distribuiranih aplikacijah. Interakcija s storitvami poteka z izmenjavo sporočil SOAP (angl. Simple Object Access Protocol) na način, ki je predpisan s specifikacijo spletne storitve. Pri tem se običajno uporabljata protokol HTML in serializacija podatkovnih struktur v obliko XML. Slika 2.1 prikazuje medsebojno povezanost tehnologij XML, SOAP in spletnih storitev.



Slika 2.1: Arhitektura spletnih storitev [7]

2.6.1 Komunikacijski protokol za izmenjavo sporočil SOAP

SOAP definira komunikacijski protokol za izmenjavo sporočil med različnimi aplikacijami in različnimi operacijskimi sistemi. Pri tem izkorišča različne transportne protokole (npr. HTTP, SMTP, TCP, JMS). SOAP je sestavljen iz ovojnice, ki je ločena na dva dela. Prvi del je glava z informacijami o avtentikaciji, kodiranju podatkov in načinu obdelave sporočil; drugi del ovojnice vsebuje sporočilo, ki je opisano v jeziku WSDL (angl. Web Services Description Language).

2.6.2 Jezik za opis spletnih storitev WSDL

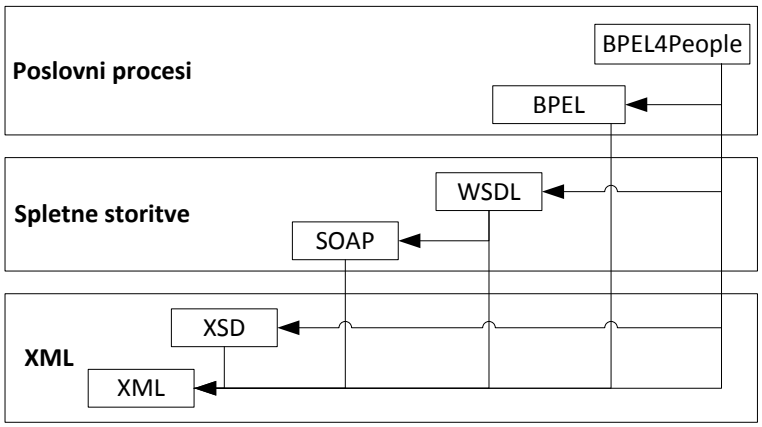
Dokument, napisan v jeziku WSDL, podrobno opiše spletne storitve in tako definira vse potrebne informacije za njihovo uporabo. Specifikacija WSDL definira naslednje elemente:

- podroben opis spletne storitve,
- metode in parametre,
- način lociranja in dostopa do spletne storitve.

Ponudnik svojo storitev definira z dokumentom WSDL in jo objavi v registru spletnih storitev UDDI (angl. Universal Description, Discovery and Integration). Odjemalec storitev locira s poizvedbo preko vmesnika UDDI in pridobi informacijo, kako lahko s storitvijo komunicira [30]. Del datoteke WSDL pove, kakšen je format vhodnih in izhodnih sporočil. Tako ima odjemalec na voljo dovolj informacij, da lahko pošlje ustrezno vhodno sporočilo. Če je klic uspešno izveden, ponudnik vrne ustrezen odgovor.

2.7 Izvršljiv jezik BPEL 2.0

Zadnji vezni člen, ki povezuje vse do sedaj naštete tehnologije v smiselno celoto, je izvršljiv jezik BPEL. Na enostaven in fleksibilen način omogoča kompozicijo in orkestracijo spletnih storitev v poslovni proces. Poslovni proces v jeziku BPEL definiramo s pomočjo gradnikov, ki jih imenujemo aktivnosti. S sestavljanjem manjših aktivnosti v kompleksne tokove opišemo korake procesa. Medsebojna odvisnost tehnologij na podlagi jezika XML je prikazana na sliki 2.2.



Slika 2.2: Graf odvisnosti med XML in BPEL4People

Osnovne aktivnosti so enostavni elementi, ki jih definira jezik BPEL, in opravljajo več nalog. Proženje drugih spletnih storitev, čakanje na vhodno zahtevo, možnost manipulacije z vrednostmi nad spremenljivkami in odkrivanje ter upravljanje z napakami in izjemami. Za združevanje osnovnih aktivnosti v kompleksnejšo logiko jezik BPEL podpira tudi sestavljene aktivnosti. Omogoča zaporedno in vzporedno izvajanje aktivnosti, implementacijo odločitvene logike glede na izpolnitev pogojev ter uporabo zank. BPEL za svoje izvajanje potrebuje strežniško infrastrukturo, ki skrbi za instance procesov in njihova stanja. S tem je centraliziran nadzor nad procesi, kar poenostavi vzdrževanje [8].

2.7.1 Razširitev jezika BPEL s človeško aktivnostjo

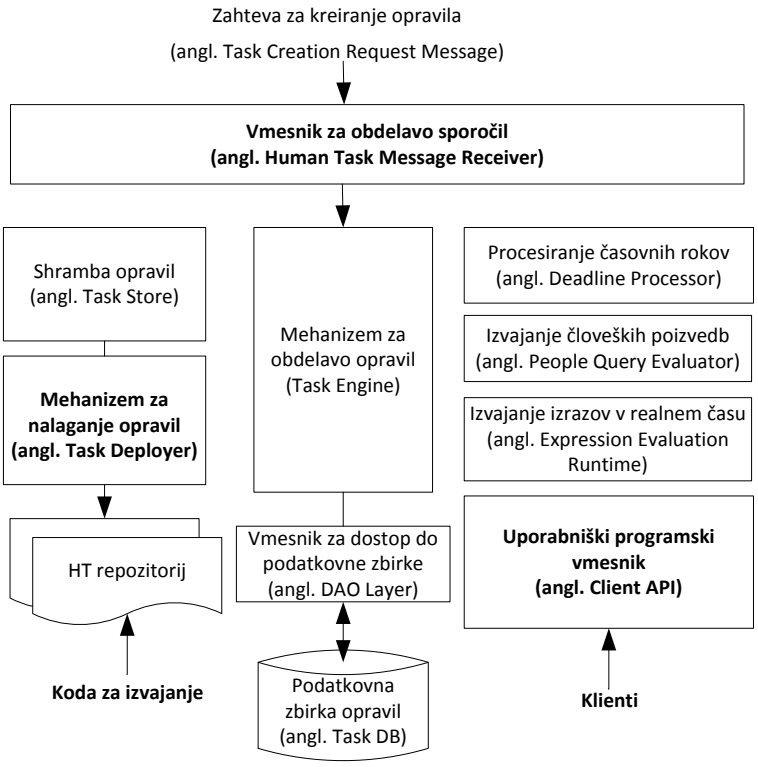
Eden od pomembnih novih konstruktov, ki se pojavijo v verziji jezika BPEL 2.0, je aktivnost <*ExtensionActivity*> [4]. Uporablja se kot ovojnica za vključevanje novih tipov aktivnosti v proces, ki v osnovi niso podprti v jeziku BPEL. Vsebina tega elementa ne sme biti prazna in mora vsebovati en tip aktivnosti, ki izpostavlja svoje atribute in standardne elemente. V kodi 2.1 je nova aktivnost imenovana z *anyElementQName*.

Koda 2.1: Definiranje novega tipa aktivnosti v jeziku BPEL 2.0. [4]

```
<extensionActivity>
  <anyElementQName standard-attributes>
    standard-elements
  </anyElementQName>
</extensionActivity>
```

2.8 Upravitelj človeških opravil (angl. Human Task Manager)

Upravitelj človeških opravil (v nadaljevanju upravitelj opravil) je samostojna programska komponenta, izpostavljena zunanjemu svetu na podlagi spletnih storitev. Njegova vloga je sprejemati zahteve za kreiranje opravil, skrbeti za njihovo instanco, življenjski cikel in pripadajoči izvajalni kontekst. Tipična arhitektura sistema za upravljanje s človeškimi opravili je prikazana na sliki 2.3.



Slika 2.3: Arhitektura upravitelja opravil [34]

Programsko kodo za izvajanje opravil in vse pomožne datoteke v obliki paketa odložimo v repozitorij, do katerega dostopa upravitelj opravil. Datoteke obdela program za nalaganje opravil, ki shrani opravilo v shrambo, in vzpostavi spletne storitve za komunikacijo z opravilom.

Vhodne zahteve za kreiranje opravila sprejme in obdela vmesnik za obdelavo vhodnih sporočil. Na podlagi vhodne zahteve upravitelj opravil najde ustrezno definicijo opravila v shrambi opravil in kreira novo instanco opravila. Med izvajanjem opravila so aktivni tudi različni moduli, ki zagotavljajo pravilno izvajanje opravil. Skrbijo za obdelavo časovnih rokov opravila, izvajajo poizvedbe po uporabniških vlogah in razrešujejo izraze v realnem času.

Za trajnejše hranjenje informacij o opravilih sistem dostopa do podatkovne zbirke.

Eden izmed pomembnejših modulov je programski vmesnik za podporo odjemalcem, ki omogoča dostop do informacij o stanju opravil zunanjim sistemom oziroma odjemalcem. Bolj podrobno ga bomo razložili v podpoglavju 5.8.

Poglavje 3

Razširitev izvedljivih poslovnih procesov s človeško aktivnostjo

3.1 Motivacija za uporabo človeške aktivnosti v poslovnem procesu

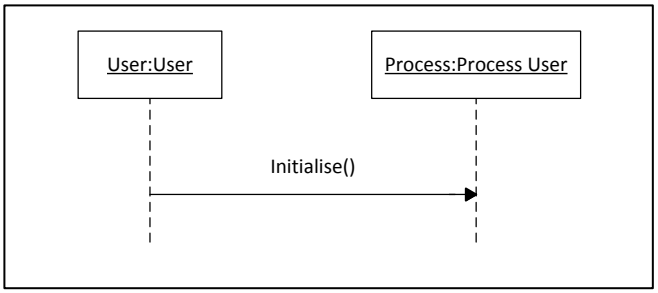
Kadar se v poslovni proces vključujejo ljudje, je logična posledica, da bo v neki točki izvajanja procesa človek moral izvesti določene aktivnosti. V splošnem vsem procesom ne ustreza, da se človeška aktivnost zaveda procesa, v katerem nastopa. V tem primeru uporabimo drugačen pristop. Če človeške aktivnosti vključimo v proces kot spletne storitve, lahko s stališča procesa enostavno preklapljamo med avtomatskimi storitvami in storitvami, ki jih izvede človek. Ob tem ni treba spreminjati definicije izvedljivega procesa, saj v obeh primerih ostane način uporabe storitev enak.
Ker ljudje igrajo pomembno vlogo v večini primerov poslovnih aplikacij, ki upoštevajo arhitekturo SOA, je iz njenega stališča ključnega pomena, da se tudi interakcija človeka s procesom modelira s spletnimi storitvami.

3.2 Uporabniška interakcija s procesom

Za boljše razumevanje potrebe po formalizaciji integracije uporabnika v procese moramo naprej analizirati različne uporabniške interakcije s poslovnimi procesi. Načine vključevanja uporabnika v izvedljivi poslovni proces razdelimo na tri osnovne skupine:

Uporabnik ⇒ Proces

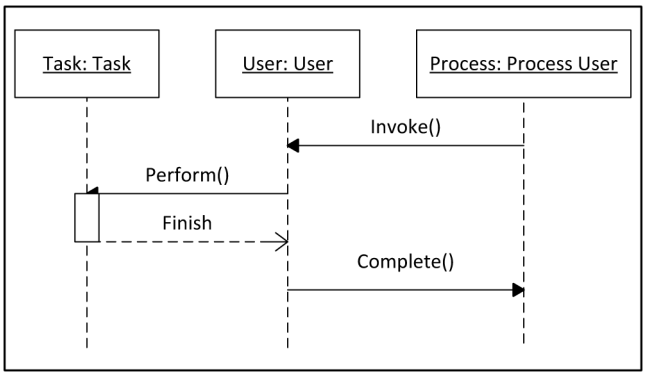
Pri enosmerni interakciji instanco procesa kreira uporabnik (angl. instantiation). Proces med izvajanjem obdeluje podatke, ki jih je na vhodu podal uporabnik ali pa so nastali kot rezultat izvedenih opravil znotraj procesa.



Slika 3.1: Sekvenčni diagram kreiranja instance procesa

Uporabnik ⇔ Proces

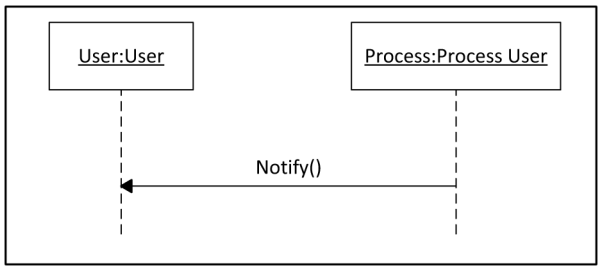
Pri dvosmerni interakciji proces obvesti uporabnika, da so potrebne dodatne informacije za nadaljevanje izvajanja. Uporabnik tipično izvede opravilo in priskrbi podatke. Osnovni podatek je tudi informacija, ali je opravilo odobreno ali zavrnjeno. Rezultat izvedenega opravila se uporabi v nadaljevanju procesa. Tak način interakcije s procesom imenujemo *uporabniška aktivnost*.



Slika 3.2: Sekvenčni diagram izvajanja uporabniške aktivnosti

Uporabnik ⇐ Proces

Enosmerno interakcijo, pri kateri je pobudnik proces, imenujemo obvestilo (angl. notification). Uporabnik v tem primeru nima vpliva na nadaljevanje procesa. Ko je obvestilo kreirano, se izvajanje procesa brez čakanja nadaljuje.



Slika 3.3: Sekvenčni diagram kreiranja obvestila

3.3 Aktivnost za razširitev jezika BPEL

Specifikacija BPEL4People je zastavljena tako, da temelji na specifikaciji BPEL. Torej lahko uporabljamo vse konstrukte jezika BPEL. Ko želimo v definicijo procesa vključiti novo aktivnost, ki v osnovi ni podprta v jeziku BPEL, moramo najprej definirati njen imenski prostor. To omogoča prepoznavo konstruktov naše nove razširitve. Naslednji primer kode 3.1 prikazuje, kako definiramo imenski prostor nove razširitve.

Koda 3.1: Definiranje imenskega prostora razširitve v jeziku BPEL 2.0

```
<bpel:process ...> ...
  <bpel:extensions>
  <bpel:extension namespace="anyURI" mustUnderstand="yes|no" />
  </bpel:extensions> ...
</bpel:process>
```

Tako strežnik BPS (angl. Business Process Server) prepozna našo novo aktivnost in zna z njo tudi upravljati. Atribut *mustUnderstand* z vrednostjo *"yes"* določa, da se pred izvajanjem nove aktivnosti obvezno preveri, ali je v določenem imenskem prostoru registriran tudi programski paket, ki zna obravnavati to razširitev. Če ta ni registriran, mehanizem zavrne izvajanje procesa. Če je vrednost atributa enaka *"no"*, se v primeru neobstoja programskega paketa izvajanje nadaljuje s prazno aktivnostjo <*empty*>.

3.4 Registracija novega tipa aktivnosti na strežnik BPS

Strežnik BPS lahko izvaja novo aktivnost samo, če ima dostop do programskega paketa in le-ta podpira vse funkcionalnosti, ki naj bi jih nova aktivnost vsebovala. Postopek oziroma primer implementacije programskega paketa za podporo novi aktivnosti BPEL je določen v dokumentaciji strežniške infrastrukture, ki podpira izvajanje jezika BPEL 2.0. Razlikuje se med različnimi ponudniki strežniške infrastrukture za podporo izvedljivim procesom v jeziku

BPEL. Razvijalcu je na voljo vmesnik v programski kodi, ki ga poljubno implementira in njegovo izvedljivo kodo kot vtičnik registrira v sistem. Naslednja psevdokoda 3.2 prikazuje implementacijo novega tipa aktivnosti:

Koda 3.2: Psevdokoda implementacije novega tipa aktivnosti

```
1  public class CustomExtendedActivity implements ExtentedActivityInterface {
2  //naša nova aktivnost
3    @Override
4    public void validate() {}
5    @Override
6    /* @extensionContext: izvajalni kontekst za nove tipe aktivnosti
7    * @param element: <extensionActivity><b4bElement>... */
8    public void run(Object extensionContext, Element element) throws
     FaultException {
9      //implementacija aktivnosti
10     ExtensionContext EC = (ExtensionContext)extensionContext;
11     //kreiramo aktivnost na podlagi elementa
12     EC.invokeActivity(element);
13     //aktivnost zaključimo uspešno
14     EC.complete();
15     //ali aktivnost zaključimo neuspešno
16     Exception e = new Exception("Custom Exception");
17     EC.completeWithFault(e);
18   }
19 }
20 public class CustomExtendedActivityBundle extends AbstractExtensionBundle {
21 //našo aktivnost s tem objektom registriramo
22  public static final String NameSpace =
    "http://diplomska/extensions/b4bElement";
23  @Override
24  public void registerExtensionActivities()  {
25   registerExtensionActivity("b4bElement",CustomExtendedActivity.Class);
26  }
27 }
```

3.5 Človeška opravila (angl. Human Tasks)

Človeška opravila so množica storitev, ki ni implementirana s programsko kodo, ampak za njihovo implementacijo oziroma izvedbo poskrbi uporabnik. Izpostavljena so preko dveh vmesnikov:

- Prvi vmesnik izpostavlja opravilo kot storitev in je namenjen integraciji opravila v poslovne procese na podlagi koordinacijskega protokola (na primer opravilo odobritve dopusta zaposlenemu).

- Drugi vmesnik nudi deležnikom opravila vse potrebne metode za upravljanje (na primer pridobiti seznam opravil).

Da lahko opravila izvaja človek, morajo biti opravilu dodeljene različne uporabniške vloge, preko katerih se končnemu uporabniku dovoli točno določen nivo sodelovanja. Opravilo je mogoče opremiti z metapodatki, ki opišejo, kako se opravilo lahko izrisuje na različnih napravah in aplikacijah ter tako omogočajo integracijo z različnimi vrstami programske opreme.

Ker se opravila praviloma izvajajo dlje časa, je zagotovljen tudi mehanizem za obravnavo časovnih in pogojnih omejitev, ki prožijo eskalacijske aktivnosti. Ena od teh aktivnosti je poseben tip opravila - obvestilo, ki za razliko od opravila ne potrebuje odzivnosti uporabnika (na primer obvestilo o preteku določenega obdobja za izvedbo opravila).

Dva pomembna cilja implementacije opravil v obliki storitev na podlagi enotne specifikacije sta:

- Prenosljivost (angl. Portability) - Omogočiti prenos obstoječih opravil in obvestil iz enega sistema v drug sistem za izvajanje človeških opravil pri drugem ponudniku.

- Interoperabilnost (angl. Interoperability) - Z natančno definiranimi vmesniki za izmenjavo sporočil in protokoli omogočiti povezljivost in interakcijo različnih komponent različnih ponudnikov.

3.5.1 Dodeljevanje opravil (angl. Assignment)

Deležniki opravila in njihove naloge oziroma omejitve so določene s človeškimi vlogami (angl. human roles). Generične človeške vloge so podrobneje opisane v podpoglavju 4.4. Pri kreiranju opravila se za vsako od generičnih vlog izvede uporabniška poizvedba (angl. people query). Če uporabniške poizvedbe ne uspejo, obravnavamo njihov rezultat kot prazno množico. Obvezno se mora v okviru opravila definirati seznam potencialnih lastnikov, ki pa je lahko prazen. Za definiranje uporabniških vlog specifikacija predpisuje element *<htd:peopleAssignments>*.

V praksi se pojavi potreba po zahtevnejših pravilih dodeljevanja opravila uporabnikom, zato imamo na voljo dva mehanizma za dodeljevanje potencialnih lastnikov:

- **Vzporedno (angl. Parallel)**: Opravilo zahteva več hkratnih lastnikov. Za vsakega potencialnega lastnika se kreira podopravilo, ki se izvaja vzporedno.
- **Zaporedno (angl. Sequential)**: Opravilo zahteva več hkratnih lastnikov. Za vsakega potencialnega lastnika se kreira podopravilo, ki se izvaja zaporedno.

Oba mehanizma lahko poljubno gnezdimo in tako gradimo kompleksnejše modele dodeljevanja opravil.

3.5.2 Nominacije (angl. Nominations)

Če je množica potencialnih lastnikov prazna, se opravilo vseeno kreira. Za uspešen prevzem lastništva in izvedbo opravila pa potrebuje opravilo vsaj enega potencialnega lastnika. V tem primeru se izvede postopek nominacije. Nominacijo izvede poslovni administrator opravila (angl. task's business administrator) z uporabo administrativne metode nominacije (podpoglavje 5.8.4). Nominirani uporabniki prevzamejo vlogo potencialnih lastnikov opravila. Ob kreiranju obvestila nominirani uporabniki prevzamejo vlogo prejemnikov obvestila.

3.5.3 Eskalacije opravil (angl. Escalations)

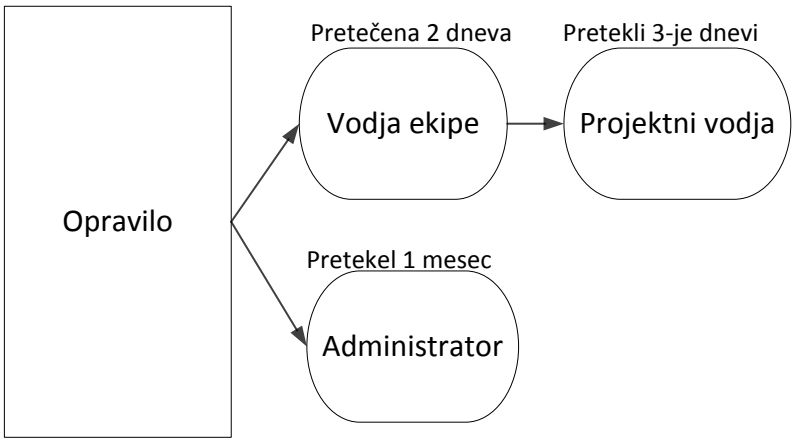
Eskalacije opravil imenujemo skupek akcij, ki se izvedejo ob izpolnitvi vnaprej določenih pogojev. Vsako opravilo, ki je modelirano kot končno opravilo, ima predpisane časovne omejitve (npr. oseba, ki bi morala v okviru opravila opraviti delo, lahko zaradi bolezni ali prevelike količine dela povzroči kritično podaljšanje časa izvajanja opravila). Z eskalacijami take omejitve opišemo tako, da s pravili določimo pogoje in definiramo množico deležnikov, ki ob izpolnitvi pogojev prejmejo obvestilo ali postanejo potencialni lastniki opravila. Eskalacije v opravilu definiramo z elementom <*htd:escalation*>.

Pogoja za proženje eskalacije sta:

1. Prekoračitev časovne omejitve za izvedbo, s katero je definirana eskalacija.

2. Izpolnjenost dodatnega pogoja.

Eskalacije so v opravilu podprte skozi celoten njegov življenjski cikel. Za aktivnost je možno definirati poljubno mnogo eskalacij, ki jih lahko združujemo v eno ali več verig. Slika 3.4 prikazuje praktičen primer opravila, ki ima definirani dve verigi eskalacij. V prvi verigi se preveri, ali je opravilo končano po dveh dneh. Če pogoj ni izpolnjen, se kreira obvestilo vodji ekipe. Naslednja kontrola se izvede tri dni po prvem preverjanju oziroma pet dni od kreiranja opravila. Če opravilo ni končano, se pošlje obvestilo projektnemu vodji. Druga veriga se izvaja vzporedno s prvo verigo in se po enem mesecu preveri, ali je opravilo končano. Če pogoj ni izpolnjen, se obvesti administratorja.



Slika 3.4: Praktični primer verige eskalacij

3.5.4 Upravitelj obvestil (angl. Notification manager)

Upravitelj obvestil nadzira opravila in proži eskalacije. Če so pogoji za eskalacijo izpolnjeni, upravitelj obvestil ustvari novo instanco eskalacijskega opravila in pošlje obvestilo vsem vnaprej določenim prejemnikom. Če se nadrejeno opravilo zaključi, upravitelj poskrbi, da se zaključi tudi eskalacijsko opravilo. Eskalacijska opravila ne potrebujejo človeške aktivnosti in jih popolnoma nadzoruje upravitelj obvestil.

3.5.5 Notifikacije (angl. Notifications)

Ena od akcij eskalacije je obveščanje deležnikov opravila o stanju opravil. Pri tem se uporablja mehanizem obvestil. Ko je izpolnjen časovni pogoj, določen v okviru eskalacije, se izvede kontrola, ali je opravilo v časovnem okviru doseglo pričakovano stanje. Če pričakovano stanje ni bilo doseženo, se proži obvestilo. Notifikacije v opravilu definiramo z elementom <*htd:notification*>.

Primer kode 3.3 definira časovno omejitev na opravilu. Če opravila ne začne izvajati nihče v roku dveh dni in je prioriteta zahteve za popravek

programa višja ali enaka 3, se pošlje obvestilo vodji.

Koda 3.3: Primer notifikacije ob eskalaciji

```
<htd:startDeadline>
  <htd:for>PT2D</htd:for>
  <htd:escalation name="obvestiVodjo">
    <htd:condition>
      <![CDATA[
        htd:getInput("ZahtevaZaDodelavoPrograma")/prioriteta <= 3
      ]]>
    </htd:condition>
    <htd:localNotification reference="tns:ObvestiloVodji">
      <htd:documentation>
        Obvestimo vodjo.
      </htd:documentation>
      <htd:peopleAssignments>
        <htd:recipients>
          <htd:from logicalPeopleGroup="vodje">
            <htd:argument name="role"> vodje </htd:argument>
          </htd:from>
        </htd:recipients>
      </htd:peopleAssignments>
    </htd:localNotification>
  </htd:escalation>
</htd:startDeadline>
```

3.5.6 Ponovna dodelitev (angl. Reassignment)

Še ena od akcij eskalacije je ponovna dodelitev lastništva nad opravilom. Uporablja se tedaj, ko je potrebno ob eskalaciji zamenjati potencialnega lastnika opravila. Ponovno dodelitev lastništva v opravilu definiramo z elementom *<htd:reassignment>*.

Primer kode 3.4 definira časovno omejitev na opravilu. Če referent A, ki je prevzel opravilo, tega ne opravi v predpisanem času petih dni, se opravilo dodeli drugemu potencialnemu lastniku. Referent B, ki spada v skupino analitikov, lahko opravilo prevzame in s tem se omogoči uspešnejšo izvedbo naloge, definirane v okviru opravila.

Koda 3.4: Primer predaje lastništva nad opravilom ob eskalaciji

```
<htd:completionDeadline>
  <htd:for>PT5D</htd:for>
  <htd:escalation name="predajAnalitiku">
    <htd:reassignment>
      <htd:documentation>
        Če je preteklo 5 dni, predamo opravilo analitiku.
      </htd:documentation>
      <htd:potentialOwners>
        <htd:from logicalPeopleGroup="analitiki">
          <htd:argument name="role"> analitiki </htd:argument>
        </htd:from>
      </htd:potentialOwners>
    </htd:reassignment>
  </htd:escalation>
</htd:completionDeadline>
```

Na opravilu lahko definiramo poljubno mnogo ekalacij, ki vsebujejo element za ponovno dodelitev opravila. Če se hkrati proži več eskalacij, se izvede prva dodelitev opravila po abecednem vrstnem redu. Opravilo po dodelitvi preide v stanje pripravljenosti in je tako na voljo potencialnim lastnikom za prevzem lastništva.

3.6 Scenariji uporabe človeških opravil

3.6.1 Verižno izvajanje (angl. Chained execution)

Verižno izvajanje opravil je eden od načinov izvajanja logično povezanih opravil. Dve opravili sta lahko logično povezani, kar pomeni, da je njuno zaporedno izvajanje iz uporabniškega vidika smiselno. Velikokrat zaporedje opravil ni vnaprej določeno in je znano šele med izvajanjem.

Ko lastnik prvega opravila to uspešno zaključi, samodejno zahteva lastništvo nad naslednjim logično povezanim opravilom v okviru iste instance procesa (angl. complete and claim next task). Če pri sekvenčnem izvajanju povezanih opravil sodeluje ista oseba, ima občutek, da jo skozi proces

reševanja opravil vodi "čarovnik". V primeru, da je deležnikov več, se opravilo verižno izvaja preko seznama opravil.

3.6.2 Scenarij delitve dolžnosti (angl. 4-Eyes Principle)

Delitev dolžnosti je pogost scenarij v poslovnih procesih, kjer se mora poslovna odločitev sprejeti v dveh korakih. Oba koraka sta dodeljena isti skupini ljudi, vendar morata odločitev sprejeti dve različni osebi. Največkrat je vzrok temu varnost, zato uporabniki ne smejo vedeti drug za drugega, da ne bi prišlo do neželenega vpliva na odločitev (npr. anketiranje). V manj zahtevnih primerih gre lahko le za drugo mnenje ali zgolj za primerjavo rezultatov.

Ob večnivojski delitvi dolžnosti obstaja v sistemu več enakih opravil in vsa imajo določenega istega potencialnega lastnika. Sistem mora znati tistega potencialnega lastnika, ki je zahteval prvo opravilo in postal dejanski lastnik, izločiti iz seznama potencialnih lastnikov vseh ostalih opravil. Torej sistem lahko dovoli potencialnemu lastniku, da zahteva samo eno od enakih opravil in pri tem informacijo o dejanskem lastniku skrije pred ostalimi potencialnimi lastniki.

Enostavno lahko sistem delitve dolžnosti uporabimo pri notranjih človeških opravilih, saj se vsa zavedajo izvajalnega konteksta procesa, v katerem nastopajo. Drugo opravilo lahko zato neposredno dostopa do informacij o prvem opravilu.

3.6.3 Kompozicija opravil in podopravil

Razbitje kompleksnega opravila v podopravila omogoča lažje in bolj nadzorovano reševanje problema. V okviru človeških opravil kompozicijo opišemo z elementom <*htd:composition*>. Podopravila so po lastnostih in obnašanju enaka nadrejenemu opravilu. V okviru opravila morajo vsa definirana podopravila imeti unikatno ime. Nadrejeno opravilo se lahko zaključi šele, ko

so zaključene vse njegove instance podopravil. Če uporabimo kompozicijo v človeškem opravilu, moramo definirati vsaj eno podopravilo. Podopravila lahko poljubno gnezdimo in tako zmanjšamo kompleksnost nadrejenega opravila. Na nivoju nadrejenega opravila definiramo različne načine kompozicije podopravil z naslednjimi atributi:

1. Glede na želeni vrstni red izvajanja določimo:

   - **Vzporedno (angl. Parallel)**: Podopravila se kreirajo v naključnem vrstnem redu in izvajajo vzporedno.
   - **Zaporedno (angl. Sequential)**: kreiranje podopravil poteka po vnaprej določenem zaporedju, urejenem po imenskem abecednem vrstnem redu podopravil. Predhodno podopravilo se mora zaključiti, predno se aktivira naslednja instanca podopravila. Zaporedno izvajanje opravil je prednastavljen način kreiranja.

2. Glede na želeni način kreiranja podopravil določimo:

   - **Ročno (angl. Manual)**: Podopravilo se kreira, ko to ročno potrdi dejanski lastnik nadrejenega opravila. Ročno kreiranje podopravil je prednastavljen način kreiranja opravil.
   - **Samodejno (angl. Automatic)**: Podopravila se kreirajo samodejno, ko se začne delo na nadrejenem opravilu. Ta preide v status *"v teku"* (angl. in progress).

Na opravilu lahko definiramo več različnih pogojev za dokončanje opravila oziroma podopravil. Pogoji se preverjajo imensko po abecednem vrstnem redu. Ob kompoziciji se ob prvem izpolnjenem pogoju trenutno podopravilo zaključi in vsa ostala opravila se preskočijo. Podprta sta dva načina zaključevanja opravil:

- **Ročno (angl. Manual)**: Dejanski lastnik mora eksplicitno končati opravilo, ko so izpolnjeni pogoji.

- **Samodejno (angl. Automatic)**: Trenutno opravilo oziroma nadrejeno opravilo se samodejno zaključi, ko so izpolnjeni pogoji.

Ob kompoziciji opravil je pomembno tudi, kako se določi izhodni rezultat obravnavanih opravil. Če več uporabnikov sodeluje na vzporednih opravilih, je potrebno rezultate ustrezno agregirati v končni rezultat nadrejenega opravila. Agregacijo rezultatov opišemo z elementom <*htd:aggregate*> in uporabo agregacijskih metod (podpoglavje 5.8.5).

Poglavje 4

Specifikacija BPEL4People

Namen specifikacije BPEL4People je zagotoviti formalen opis konstruktov v jeziku BPEL, ki nudijo podporo množici scenarijev in vključujejo ljudi v izvedljive poslovne procese.

4.1 Uporabniške aktivnosti

Za integracijo ljudi v izvedljive poslovne procese potrebujemo poseben tip aktivnosti BPEL. To novo aktivnost imenujemo uporabniška aktivnost. Definiramo jo z elementom *<b4p:peopleActivity>*. Primer posplošene kode:

Koda 4.1: Primer uporabe uporabniške aktivnosti v jeziku BPEL 2.0

```
<bpel:process
xmlns:b4p="http://docs.oasis-open.org/ns/bpel4people/bpel4people/200803"
xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803">
  ...
<bpel:extensions>
  <bpel:extension
  namespace="http://docs.oasis-open.org/ns/bpel4people/bpel4people/200803"
  mustUnderstand="yes"/>
  <bpel:extension
  namespace="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
  mustUnderstand="yes"/>
</bpel:extensions>
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/" ... \>
```

```
  ...
<bpel:extensionActivity>
  <b4p:peopleActivity name="NCName" ...> ... </b4p:peopleActivity>
</bpel:extensionActivity>
  ...
</bpel:process>
```

4.2 Lastnosti uporabniške aktivnosti

Pomembna in ključna lastnost uporabniške aktivnosti - v primerjavi z drugimi aktivnostmi v jeziku BPEL - je v načinu njene implementacije. Uporabniška aktivnost ne rešuje poslovnega problema v obliki programske kode. Dejansko se med izvajanjem uporabniške aktivnosti pokliče metodo spletne storitve, ki predstavlja povezavo med procesom in uporabnikom. Spletno storitev izpostavlja t. i. **človeško opravilo** (poglavje 5) in zanj skrbi upravitelj opravil. Deležniku procesa oziroma opravila je omogočen dostop do opravil preko **seznama opravil**.

Uporabniška aktivnost v svoji definiciji predvideva atribute:

- **Vhodna in izhodna spremenljivka**: uporabljata se za izmenjavo vhodnih in izhodnih sporočil pri klicu metode spletne storitve.
- Z atributom **isSkipable** določimo, ali je mogoče kreirano opravilo preskočiti in nadaljevati z izvajanjem procesa.
- Z elementom **b4p:scheduledActions** določimo časovne okvire, v katerih se mora stanje opravil spremeniti. Vplivamo lahko na rok veljavnosti opravila ali končni rok za aktivacijo opravila.

4.3 Integracija uporabniške aktivnosti v proces BPEL

Glede na proces in uporabniško aktivnost razlikujemo dva glavna tipa opravil, in sicer notranja in samostojna opravila [4]. Njune pomembne razlike so navedene v tabeli 4.1.

Notranje človeško opravilo (angl. Inline human task) se zaveda procesa BPEL in proces se zaveda opravila. Opravilo ima dostop do izvajalnega konteksta procesa. Uporablja lahko spremenljivke, lastnosti in podatke procesa. Proces opravilo kreira, ravno tako določa njegov življenjski cikel in lahko izbriše njegovo instanco. Takšno opravilo ni na voljo za ponovno uporabo drugim procesom ali aktivnostim. Notranja človeška opravila je smiselno uporabiti v naslednjih primerih:

- kadar želimo neposredno dostopati do izvajalnega konteksta procesa,
- kadar potrebujemo v opravilo prenesti rezultat logike procesa in se s tem rešiti eksplicitnega modeliranja vhodnih sporočil opravila,
- kadar želimo izvajati administrativna opravila,
- kadar želimo določiti avtorizacijo za točno določeno uporabniško aktivnost,
- kadar potrebujemo mehanizem delitve dolžnosti (podpoglavje 3.6.2).

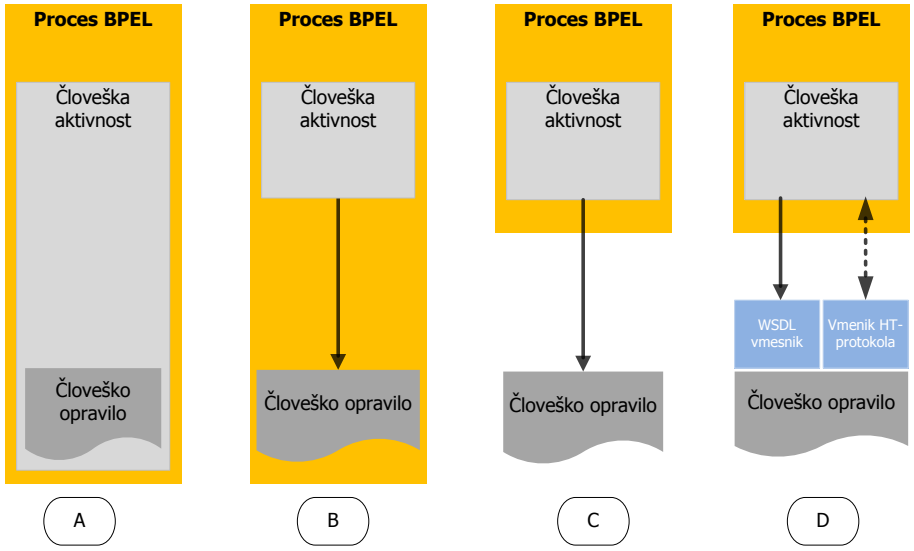
Samostojno človeško opravilo (angl. Standalone human task) je neodvisno od procesa BPEL in sledi servisno orientirani arhitekturi (SOA). Preko vnaprej definiranega vmesnika je dostopno tudi drugim komponentam in definira vhodna ter izhodna sporočila. Njegov življenjski cikel ni odvisen od procesa, v katerem nastopa, in rezultat opravila je na voljo tudi potem, ko proces ne obstaja več. Ker opravilo modeliramo ločeno od procesa, ga lahko ponovno uporabimo v različnih procesih. Samostojna človeška opravila je smiselno uporabiti v naslednjih primerih:

- kadar opravilo od procesa ne potrebuje nobene informacije,
- kadar opravilo v našem sistemu predstavlja zgolj samo dodatno storitev,
- kadar želimo pustiti odprto možnost, da se spremeni opravilo, vendar pri tem ne želimo spreminjati komponente, ki opravilo uporablja.

Tabela 4.1: Lastnosti notranjih in samostojnih človeških opravil

Karakteristike	**Notranje človeško opravilo**	**Samostojno človeško opravilo**
tip opravila	sodelujoča opravila, izvorna opravila, administrirativna opravila	sodelujoča opravila, izvorna opravila, čista človeška opravila
partner v interakciji	proces BPEL	katera koli storitev (komponenta SOA)
možnost zamenjave z drugo komponento SOA	ne	da
opravilo ima dostop do izvajalnega okolja procesa	da	ne
življenjski cikel	odvisen od procesa	neodvisen od procesa
mogoče ponovno uporabiti	ne	da

Na sliki 4.1 so glede na način, kako definiramo človeško opravilo, prikazani štirje modeli integracije notranjih in samostojnih človeških opravil v izvedljivi poslovni proces [4, 12].



Slika 4.1: Primeri integracije opravila v proces BPEL [4]

Primer kode 4.2 prikazuje notranje človeško opravilo, ki je deklarirano v okviru uporabniške aktivnosti znotraj procesa BPEL. Prednost tega načina je neposreden dostop do spremenljivk procesa.

Koda 4.2: Prvi primer uporabe uporabniške aktivnosti v jeziku BPEL 2.0.

```
<bpel:extensionActivity>
  <b4p:peopleActivity ...> <!-- primer A -->
    <htd:task />
  </b4p:peopleActivity>
</bpel:extensionActivity>
```

Primer kode 4.3 prikazuje uporabo opravila, ki je definirano izven uporabniške aktivnosti, vendar še vedno v kontekstu procesa. Glede na ponovno uporabo je to dobro, saj lahko opravilo nastopa v več človeških aktivnostih znotraj istega procesa.

Koda 4.3: Drugi primer uporabe uporabniške aktivnosti v jeziku BPEL 2.0.

```
<bpel:extensionActivity>
  <b4p:peopleActivity ...> <!-- primer B -->
    <b4p:localTask reference="tns:notranjeOpravilo"/>
    <!-- opravilo je definirano v tem procesu -->
  </b4p:peopleActivity>
</bpel:extensionActivity>
```

Primer kode 4.4 je zelo podoben drugemu. Spada v skupino samostojnih človeških opravil, ker je definirano zunaj procesa BPEL in je zato od njega neodvisno.

Koda 4.4: Tretji primer uporabe uporabniške aktivnosti v jeziku BPEL 2.0.

```
<bpel:extensionActivity>
  <b4p:peopleActivity ...> <!-- primer C -->
    <b4p:localTask reference="tns:zunanjeOpravilo"/>
    <!-- opravilo ni definirano v tem dokumentu -->
  </b4p:peopleActivity>
</bpel:extensionActivity>
```

Primer kode 4.5 prikazuje uporabo samostojnega človeškega opravila, vendar v drugem okolju. Opravilo izpostavlja svoj vmesnik, s katerim komuniciramo s pomočjo protokolov spletnih storitev in s predpisanim koordinacijskim protokolom WS-HumanTask.

Koda 4.5: Četrti primer uporabe uporabniške aktivnosti v jeziku BPEL 2.0.

```
<bpel:extensionActivity>
  <b4p:peopleActivity ...> <!-- primer D -->
    <b4p:remoteTask ... />
  </b4p:peopleActivity>
</bpel:extensionActivity>
```

4.4 Človeške vloge (angl. Generic human roles)

Pri vsakem procesu lahko nastopajo različne uporabniške vloge. Če smo natančnejši, lahko uporabnik že znotraj enega procesa prevzame različne uporabniške vloge. Na primer isti referent lahko znotraj enega poslovnega procesa opravlja vsebinsko različno delo glede na trenutno fazo procesa. Medsebojno odvisnost uporabnika, procesa in opravila formaliziramo z generičnimi človeškimi vlogami. Vloge natančno definirajo, kakšen vpliv ima uporabnik ali skupina uporabnikov na instanco procesa oziroma opravila in kakšne akcije lahko izvajajo. Ločimo naslednje človeške vloge:

- **Iniciator procesa** (angl. Process initiator) je oseba, odgovorna za kreirano instanco procesa. Iniciatorja lahko določi sistem samodejno, lahko pa ga definiramo sami. Hkrati ima lahko vlogo iniciatorja procesa samo ena oseba.

- **Deležniki procesa** (angl. Process stakeholders) izvajajo nadzor nad procesom (dodajajo priloge, posredujejo opravila). Če deležnik procesa ni določen, to vlogo prevzame iniciator procesa. Ko je instanca procesa kreirana, se zahteva vsaj en deležnik procesa.

- **Iniciator opravila** (angl. Task initiator) je oseba, odgovorna za kreirano instanco opravila. Hkrati ima lahko vlogo iniciatorja opravila samo ena oseba. Odvisno od tega, kako je bilo opravilo kreirano, je iniciator opravila lahko definiran ali ne.

- **Deležniki opravila** (angl. Task stakeholders) izvajajo nadzor nad opravilom in imajo vpliv na potek opravila (dodajajo priloge, posredujejo opravilo, opazujejo spremembe stanj). Ko je instanca opravila kreirana, se zahteva vsaj en deležnik opravila.

- **Potencialni lastniki** (angl. Potential owners) lahko eksplicitno zahtevajo lastništvo nad instanco opravila. S tem postanejo tudi dejanski

lastniki. Predno prevzamejo lastništvo, lahko vplivajo na prioriteto opravila, dodajo priloge ali podajo komentarje. Ista oseba je lahko potencialni lastnik več opravilom. Če obstaja samo en potencialni lastnik, je ta samodejno dejanski lastnik.

- **Dejanski lastnik** (angl. Actual owner) je tisti, ki izvaja akcije nad opravilom (umakne zahtevo za lastništvo, spremeni prioriteto, posreduje opravilo, začasno ustavi ali nadaljuje z opravilom). Vlogo dejanskega lastnika opazovanega opravila ima lahko samo ena oseba hkrati. Ena oseba pa je lahko dejanski lastnik več opravilom. Dejanski lastnik se določi sistemsko in ga ni mogoče vnaprej določiti.

- **Izločeni lastniki** (angl. Excluded owners) ne morejo postati dejanski oziroma potencialni lastniki in ne morejo začeti ali rezervirati opravila. Vsi izločeni lastniki so tudi izločeni iz množice potencialnih lastnikov.

- **Prejemniki obvestil** (angl. Notification recipients) so osebe, ki prejmejo obvestilo, ko na primer med izvajanjem opravila pride do eskalacije. Obvestila ne kreira vedno opravilo, ampak ga lahko tudi proces ali zunanji sistem. Ta vloga se od potencialnih oziroma dejanskih lastnikov razlikuje v tem, da uporabniku ob prejemu obvestila ni treba izvesti nobene akcije.

- **Poslovni administratorji** (angl. Business administrators) imajo podobno vlogo in lahko izvajajo iste akcije kot deležniki procesa oziroma opravila. Razlika med njimi je v območju delovanja. Poslovni administrator deluje v območju celotne instance procesa oziroma opravila. Spremlja lahko tudi potek izvajanja obvestil. Med izvajanjem mora biti določen vsaj en poslovni administrator. Če administrator ni določen pred izvajanjem, to vlogo prevzamejo deležniki.

Koda 4.6: Definiranje človeških vlog [4]

```
<b4p:peopleAssignments>
  <b4p:processInitiator>?
    <htd:from ...>...</htd:from>
  </b4p:processInitiator>
  <b4p:processStakeholders>?
    <htd:from ...>...</htd:from>
  </b4p:processStakeholders>
  <b4p:businessAdministrators>?
    <htd:from ...>...</htd:from>
  </b4p:businessAdministrators>
</b4p:peopleAssignments>
```

Logične skupine uporabnikov (angl. Logical people groups) so, poleg literalov in izrazov, dodatna možnost za prirejanje oseb k ustrezni generični vlogi. Logično skupino definiramo s poizvedbo po organizacijski strukturi in s parametri poizvedbe. Rezultat poizvedbe je množica uporabnikov. Poizvedbe in parametri se upoštevajo med izvajanjem oz. ob kreiranju opravila, zato je njihov rezultat dinamičen. Dostop do organizacijskih struktur se izvede preko internetnega protokola za dostop do imenikov LDAP (angl. Lightweight Directory Access Protocol). Dostop je popolnoma abstrakten, medtem ko je njegova implementacija stvar zalednega strežniškega sistema.

Naslednji primer kode 4.7 prikazuje definicijo logične skupine uporabnikov z imenom “oddelek“, ki jo filtriramo glede na argument “podrocjeDela“. Rezultat so potencialni lastniki opravila, izbrani glede na delovno področje, ki je del vhodnega sporočila opravila:

Koda 4.7: Primer definicije logične skupine uporabnikov

```
<htd:logicalPeopleGroup name="oddelek">
  <htd:parameter name="podrocje" type="xsd:string" />
</htd:logicalPeopleGroup>
<htd:potentialOwners>
  <htd:from logicalPeopleGroup="oddelek">
```

```
    <htd:argument name="podrocje">
      htd:getInput("OpraviloRequest")/podrocjeDela
    </htd:argument>
  </htd:from>
</htd:potentialOwners>
```

Poglavje 5

Specifikacija WS-HumanTask

Človeško opravilo bomo definirali na podlagi specifikacije WS-HumanTask. Specifikacija predlaga podatkovne tipe in strukture za opis opravil, definira diagram stanj opravila in programski vmesnik za dostop do opravil. Definira tudi koordinacijski protokol, ki omogoča interakcijo človeških opravil v bolj servisno-orientirani arhitekturi in pri tem kontrolira avtonomijo opravil.

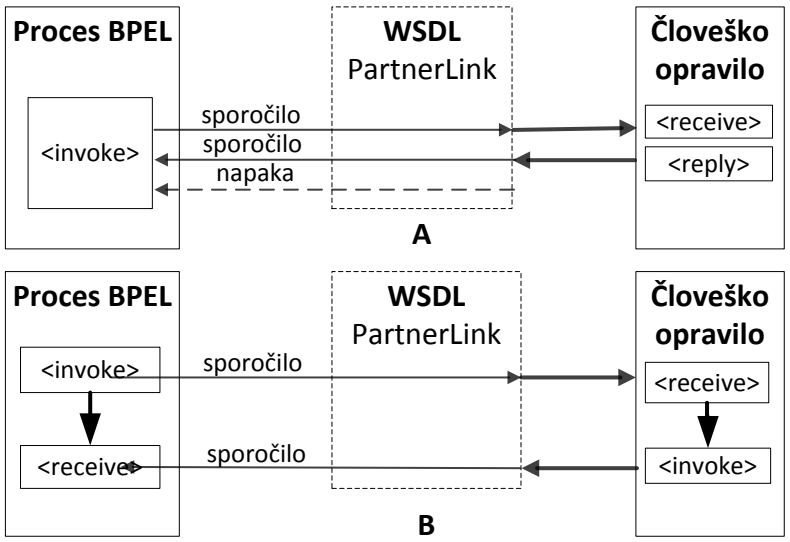
Specifikacija definira, kako izmenjujemo **vhodna in izhodna sporočila** z entitetami, ki z opravili sodelujejo v procesnem smislu. Opredeljuje **uporabniške vloge**, s katerimi se uporabniku omeji odgovornost pri dodeljevanju in prevzemanju lastništva nad opravili in tako zagotovi ustrezen konec opravila. Zaradi interakcije z opravilom v obliki spletne storitve in posledično s pomanjkanjem grafične predstavitve opravila omogoča **vključitev uporabniškega vmesnika** v opravilo. Potreben je tudi **nadzor nad stanjem opravila**, zato nam specifikacija omogoča, da lahko določimo poslovna pravila, s katerimi opredelimo izredne dogodke (npr. pretečen rok izvedbe) in nato ob tem izvedemo določene akcije (npr. spremembo prioritete ali predajo lastništva opravila).

5.1 Abstraktna definicija opravila

Lastnosti opravil v kodi abstraktno opišemo s pomočjo predpisane notacije; opišemo samo pogoje, ki vplivajo na obnašanje opravila v času izvajanja. Na primer komu bo opravilo dodeljeno v času izvajanja. Človeško opravilo je torej v svoji definiciji modeliran koncept opravila v času izvajanja. Imeti mora določeno unikatno ime. Ime opravila je pomembno zaradi dostopa do informacij o opravilu, saj predstavlja ključ za iskanje opravil v sistemu. V definiciji opravila se predpiše tudi programski vmesnik, opisan z notacijo WSDL, ki izpostavlja opravilo kot spletno storitev in nudi metodo za kreiranje. Vmesnik za kreiranje instance opravila uporabi t. i. kreator opravila (angl. task parent), ki je v praksi po navadi izvedljivi poslovni proces. Človeško opravilo vsebuje naslednje komponente:

1. **htd:interface**: Definira ime spletne storitve in metodo vmesnika WSDL, ki se pokliče ob kreiranju opravila. Glede na način uporabe ločimo dva možna vmesnika (slika 5.1):

    - V primeru A opravilo prejme vhodno sporočilo preko ene metode in takoj vrne odgovor. Definiramo samo ime storitve z elementom *portType* in metodo z atributom *operation*.
    - V primeru B opravilo prejme vhodno sporočilo preko ene metode. Ko opravilo preide v status zaključen, se odgovor asinhrono pošlje preko druge metode. Ime druge storitve in metodo definiramo z atributoma *responsePortType* oziroma *responseOperation*.



Slika 5.1: Primera dveh tipov vmesnikov WSDL na človeškem opravilu

2. **htd:priority**: Opravilu se lahko določi prioriteto, ki določa vrstni red opravljanja dela. Za primer lahko vzamemo proces odpravljanja programskih napak poslovnega informacijskega sistema, kjer je pomembno, da kritične napake odpravimo z višjo prioriteto, saj lahko pri stranki povzročijo poslovno škodo. Najvišja prioriteta je 0, najnižja prioriteta je 10 in privzeta vrednost je 5. Prioriteto se lahko s pomočjo izrazov določi tudi dinamično med izvajanjem.

3. **htd:peopleAssignments**: Med seboj enaka opravila so lahko dodeljena različnim osebam, vendar glede na definicijo opravila mora biti eno opravilo izvedeno samo enkrat. Preden uporabnik želi opraviti neko delo, mora to potrditi s prevzemom lastništva nad opravilom. S tem elementom določimo vse deležnike opravila.

4. **htd:delegation**: Ko se v izvedljivem poslovnem procesu obravnavajo uporabniške aktivnosti in posledično kreira instance opravil, upravitelj

opravil doda kreirano opravilo v delovno vrsto. Opravilo je tako na voljo v seznamu opravil vsem, ki imajo ustrezno dovoljenje za delo. S tem elementom lahko omejimo delegiranje zahtev med deležniki.

5. **htd:deadlines**: Človeške aktivnosti v poslovnih procesih se praviloma izvajajo dlje časa. Zato človeška opravila predvidevajo mehanizem za nadzor nad dolžino izvajanja opravil. V ta namen lahko v okviru posameznega opravila definiramo različne časovne okvire in akcije, ki se izvedejo po preteku časovnih intervalov (podpoglavje 5.6.4). Na primer določimo lahko maksimalni časovni okvir, v katerem mora lastništvo nad aktivnostjo prevzeti njen potencialni lastnik.

6. **htd:composition**: Delitev opravil na manjša opravila je smiselno ob kompleksnih opravilih. Ob človeških opravilih imamo možnost kompozicije podopravil. Podopravila si delijo podatke z nadrejenim opravilom, lahko se izvajajo vzporedno ali zaporedno in se kreirajo na zahtevo ali jih samodejno kreira upravitelj opravil (podpoglavje 3.6.3).

7. **htd:completionBehavior**: Določa enega ali več pogojev za zaključek opravila. Ko se izpolni vsaj en pogoj, stanje opravila preide v zaključeno.

8. **htd:possibleOutcomes**: Rezultat zaključenega opravila vpliva na potek poslovnega procesa. Opravilo lahko rezultira na več načinov, kar opišemo s tem elementom. Tipičen primer je opravilo, kjer mora odgovorna oseba odobriti ali zavrniti prošnjo za dopust podrejene osebe. V tem primeru sta rezultat lahko dve različni strukturi, ena opiše rezultat ob odobritvi in druga ob zavrnitvi.

9. **htd:presentationElements**: Smisel človeških opravil se pokaže šele, ko so njegove informacije predstavljene uporabniku. Uporabniški vmesnik je lahko implementiran v različnih tehnologijah (spletni brskalniki, Java odjemalci, SMS, elektronska pošta). Grafični elementi, predvideni v opravilu, omogočajo v strukturirani obliki izpostaviti informacije,

ki jih potrebujejo uporabniški vmesniki za predstavitev oziroma izris opravil uporabniku. Podrobneje bomo grafične elemente predstavili na praktičnem primeru v podpoglavju 6.4.

5.2 Prednosti uporabe uporabniških opravil

- **Ponovna uporaba**: Opravilo, ki smo ga definirali v enem od procesov, se lahko enostavno uporabi tudi v drugem, popolnoma različnem procesu, če je poslovna potreba po podobnem opravilu.

- **Možnost revizije**: Opravilo hrani vse pomembne informacije (npr. vhodne in izhodne informacije, kdo je opravilo izvajal in koliko časa). S temi informacijami lahko spremljamo in optimiziramo izvajanje opravila in opravljamo analitična poročila. Če so vsi poslovni procesi modelirani z uporabniškimi opravili, lahko sistem sam nadzira odzivnost sistema in ugotovi, katera opravila zahtevajo več človeških sredstev.

- **Možnost nadzora**: Opravilo je izpostavljeno opazovalcu kot spletna storitev. Vse klice spletne storitve lahko nadziramo. Nadziramo pa lahko tudi dogodke, ki se prožijo ob spremembah stanja opravila in na podlagi njih izvajamo akcije za uspešen konec opravila. S tem zagotovimo, da je delo dodeljeno pravim osebam v pravem času.

- **Fleksibilnost**: Večina poslovnih aplikacij je implementirana v obliki uporabniških vmesnikov, ki so med seboj povezani tako, da vodijo uporabnika skozi proces. Sprememba takih aplikacij je lahko postopkovno in časovno zahtevna. Opravilo je definirano v obliki datoteke XML, zato je definicijo enostavno popraviti in razširiti z dodatnimi funkcionalnostmi. Proces, modeliran z uporabniškimi opravili, lahko hitro prilagodimo poslovnim potrebam le s spremembo zaporedja klicev spletnih storitev.

5.3 Vrste človeških opravil

Glede na pobudnika opravila lahko prepoznamo štiri vrste opravil [10]:

1. **Sodelujoče opravilo** (angl. Participating task): Interakcija računalnik - človek. Poslovni proces je pobudnik opravila in čaka na uporabnikov odziv. Na primer sistem instancira opravilo in čaka na uporabnike, da ga izvedejo. Uporabnik A najde opravilo, ga prevzame in zaključi. O rezultatu je obveščen proces, ki lahko nadaljuje izvajanje.

2. **Administrativno opravilo** (angl. Administrative task): Podobno sodelujočemu opravilu. Namenjeno je reševanju tehničnih težav pri izvajanju procesa.

3. **Izvorno opravilo** (angl. Originating task): Interakcija človek - računalnik. Uporabnik je pobudnik opravila, ki proži storitve v sistemu. Na primer uporabnik A kreira opravilo. Zahteva za začetek se pošlje v sistem, ki izvede ustrezne storitve. Ko sistem zaključi izvajanje, se o tem obvesti uporabnika.

4. **Čisto človeško opravilo** (angl. Purely human task): Interakcija človek - človek. Omogoča sodelovanje uporabnikov na strukturiran in kontroliran način. Na primer uporabnik A kreira opravilo in opravi določene aktivnosti v okviru opravila ter želi, da opravi svoje aktivnosti tudi uporabnik B, zato mu prepusti opravilo. Uporabnik B poišče opravilo, ga prevzame, opravi delo in ga zaključi. O tem je obveščen tudi uporabnik A.

5.4 Življenjski cikel opravila

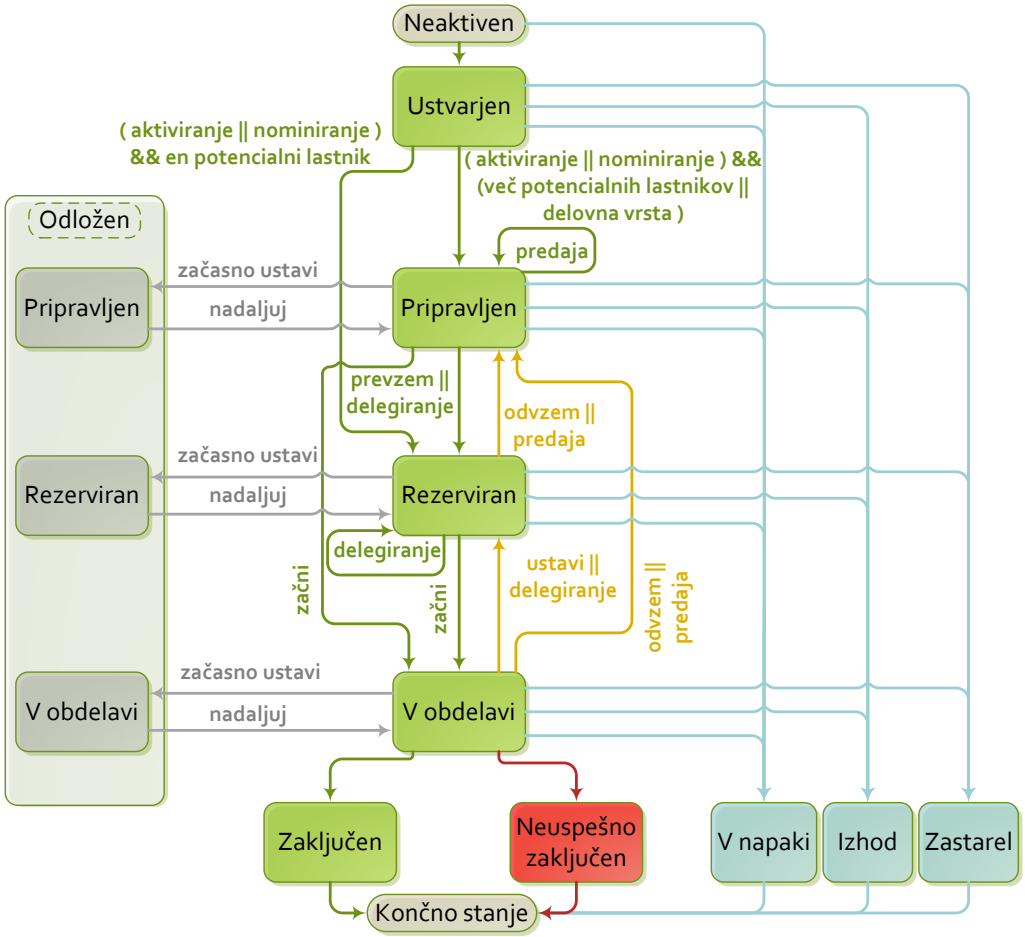
5.4.1 Diagram prehajanja stanj

Upravitelj opravil skrbi za vsa opravila med njihovim celotnim življenjskim ciklom. Opravilo je lahko po inicializaciji le v enem od možnih stanj. Vpliv

na stanja opravila imajo deležniki opravila, ki so avtorizirani za izvajanje akcij nad opravili. Razloga za prehajanje med stanji sta:

- izvedena akcija deležnika opravila (npr. uporabnik začne z izvajanjem opravila),
- spremembo zahteva koordinator opravil (npr. v primeru napake).

Med prehajanjem stanj se poslovni podatki, vezani na opravilo, ne izgubijo. Življenjski cikel opravila je prikazan na diagramu prehajanja stanj (slika 5.2). V nadaljevanju smo opisali akcije in dogodke, ki prožijo prehod med stanji.



Slika 5.2: Diagram prehajanja stanj opravila [1]

5.4.2 Normalno izvajanje opravila

Na sliki 5.2 je najpogostejši življenjski cikel opravila prikazan z zeleno barvo. Instanca novo kreiranega opravila je v začetnem stanju *ustvarjen* (angl. created). Opravilu se nastavijo lastnosti v naslednjem vrstnem redu:

1. Vhodno sporočilo se poveže z opravilom.
2. Opravilu se določi prioriteta.
3. Pripravi se seznam potencialnih lastnikov.
4. Nastavijo se še ostale lastnosti.

Uspešno kreirano opravilo prevzame koordinator opravil, mu določi koordinacijski kontekst in ga doda v svojo delovno vrsto. Opravilo lahko ostane v tem stanju, če potencialni lastniki niso bili določeni med kreiranjem. Za določitev potencialnih lastnikov med izvajanjem z nominiranjem poskrbi poslovni administrator.

Opravilo mora za uspešno izvedbo imeti vsaj enega dejanskega lastnika. Če je bil pri kreiranju določen en potencialni lastnik, postane ta tudi dejanski lastnik z avtomatskim prehodom opravila v stanje *rezerviran* (angl. reserved). Če je potencialnih lastnikov pri kreiranju več, opravilo preide v stanje *pripravljen* (angl. ready) in je tako na voljo potencialnemu lastniku, da s prevzemom postane tudi dejanski lastnik opravila. Opravilo posledično preide v stanje *rezerviran*.

S prehodom opravila v stanje *v teku* (angl. in progress) se uporabnikovo delo na opravilu dejansko začne. Uporabnik lahko delo začne le na opravilu, ki je pred tem v stanju *rezerviran* ali *pripravljen*. Ta, ko začne z delom na opravilu v stanju *pripravljen*, postane takoj tudi dejanski lastnik.

Ko je delo uspešno zaključeno, se opravilo prestavi v končno stanje *zaključen* (angl. completed). Če da se delo ne zaključi uspešno, preide opravilo v stanje *neuspešno zaključen* (angl. failed). Obe stanji v življenjskem ciklu opravila predstavljata enakovreden rezultat izvedbe. Obravnavamo ju, kot

da je opravilo uspešno izvedeno s pozitivnim oziroma z negativnim izhodnim sporočilom.

5.4.3 Neuspešno izvajanje opravila

Različni dejavniki lahko vplivajo na to, da izvajanje opravila ne poteka normalno. Na sliki 5.2 so z modro barvo prikazana stanja, v katera preide opravilo ob izrednih dogodkih.

V stanje *napake* (angl. error) lahko opravilo preide iz vsakega drugega stanja, ko pride do nepopravljive napake med izvajanjem (npr. deljenje z 0). Rezultat je v tem stanju prazno sporočilo in opravilo se obravnava kot neuspešno izvedeno.

Vpliv na opravilo ima tudi pripadajoče okolje oziroma kontekst, v katerem se opravilo izvaja (npr. opravilo je največkrat del poslovnega procesa). Okolje lahko odloči, da opravila ne potrebuje več ali da je pretekel čas izvedbe, in konča izvajanje opravila. Lahko se zaključi tudi izvajanje okolja samega. Opravilo konča v stanju *zastarel* (angl. obsolete).

Enak vpliv na opravilo imata tudi poslovni administrator in oseba, ki opravljata delo na opravilu. Če sprejmeta odločitev, da opravila ni več treba izvajati, ga lahko preskočita. Spremeni se stanje opravila in vrne prazno izhodno sporočilo, vendar se opravilo obravnava kot uspešno izvedeno. To možnost je treba omogočiti že v definiciji opravila.

5.4.4 Odvzem in predaja lastništa nad opravilom

Določena opravila za uspešno končano delo zahtevajo delitev dela med več deležniki. Ločimo naslednji vrsti izmenjave lastništva:

Delegiranje opravila (angl. Delegating)

Opravilo lahko potencialni lastnik, dejanski lastnik ali poslovni administrator delegirajo drugi osebi, ki postane hkrati tudi dejanski lastnik. Oseba se doda tudi v seznam potencialnih lastnikov, če tam še ne obstaja. Opravilo

se lahko delegira le v aktivnem stanju (*pripravljen*, *rezerviran*, *v teku*) in po uspešnem delegiranju preide v stanje *rezerviran*. Komu se opravilo lahko delegira, določimo v definiciji opravila z elementom:

```
<htd:delegation potentialDelegatees="anybody|nobody|potentialOwners|other" />
```

1. **Kdor koli (angl. Anybody)** - za novega lastnika izberemo kogar koli.
2. **Nihče (angl. Nobody)** - opravila ni dovoljeno delegirati.
3. **Potencialni lastniki (angl. Potential owners)** - novega lastnika izberemo iz množice potencialnih lastnikov na opravilu.
4. **Drugo (angl. Other)** - množico potencialnih lastnikov določimo sami.

Predaja opravila (angl. Forwarding)

Podobno kot pri delegiranju opravil tudi tukaj lahko potencialni lastnik, dejanski lastnik ali poslovni administrator predajo opravilo drugemu potencialnemu lastniku. Glavna razlika v primerjavi z delegiranjem je, da tu omejitev, komu se preda opravilo, ni tako restriktivna. Opravilo se lahko preda, če so izpolnjeni naslednji pogoji:

1. opravilo je v stanju *pripravljen*;
2. če je v stanju *rezerviran* ali *v teku*, samodejno preide v stanje *pripravljen*;
3. potencialni lastniki ne smejo biti določeni kot ena ali več uporabniških skupin, temveč kot individualni uporabniki.

Po predaji opravila sistem uporabnika, ki opravilo predaja, umakne iz seznama potencialnih lastnikov in na seznam doda uporabnika, ki mu je bilo opravilo predano.

5.4.5 Začasna zaustavitev opravila

Opravilo, ki se nahaja v enem od stanj *pripravljen*, *rezerviran* ali *v teku*, se lahko začasno zaustavi. Opravilo z ukazom *začasno zaustavi* (angl. suspend) odložimo v vrsto odloženih stanj. Opravilo v tem primeru zaradi ohranjanja informacije o predhodnem stanju preide v istoimensko podstanje. Eden od primerov, v katerem lahko to možnost izkoristimo, je ob težavi v instanci opravila. Ko opravilo popravimo in zadostimo vsem zahtevam, da se lahko opravilo ponovno izvaja, vrnemo opravilo v enako stanje kot pred ustavitvijo z ukazom *nadaljuj* (angl. resume).

5.5 Koordinacijski protokol

Koordinacija med upraviteljem opravil in kreatorjem opravila zahteva izmenjavo določenih informacij. Kreator opravila je na primer proces, v katerem opravilo sodeluje. Postopek izmenjave teh informacij določa koordinacijski protokol [4]. Koordinacija procesa in opravila je nujna zaradi več razlogov:

- Kreator opravila (proces BPEL) pošlje več zahtev za kreiranje instanc opravila upravitelju opravil. Za proces je pomembno, da se tedaj, ko se proces ne izvaja več, končajo tudi vse njegove instance opravil. To omogoča koordinacijski protokol, ki hrani informacijo o vseh opravilih in njihovih kreatorjih.

- Enak postopek je nujen za opravilo. Če se opravilo konča na strani upravitelja opravil, se mora ta informacija prenesti tudi v sodelujoči proces.

Koordinacijski kontekst je tipizirana struktura in njena shema je na voljo na [6]. Koordinacijski kontekst ustvari mehanizem za izvajanje procesa, ko ugotovi, da se mora izvesti uporabniška aktivnost. Koordinacijski kontekst se v sporočilu za kreiranje opravila prenese do upravitelja opravil. Koordi-

nacijski kontekst definira tudi protokol, po katerem si obe strani izmenjujeta koordinacijska sporočila:

- **Sporočilo o napaki (angl. Fault)**: uporabniška aktivnost prejme informacijo, da je opravilo prešlo v nepopravljivo napako,
- **Sporočilo za nadaljevanje izvajanja (angl. Skipped)**: upravitelj sporočil pošlje sporočilo uporabniški aktivnosti, če lastnik opravila odloči, da se izvajanje opravila preskoči,
- **Sporočilo za konec izvajanja (angl. Exit)**: če se opravilo ne zaključi pravočasno ali se predčasno zaključi izvajanje uporabniške aktivnosti, ta pošlje sporočilo opravilu.

5.6 Obvestila

S pomočjo opravil lahko uporabnike uspešno vključimo v izvedljivi poslovni proces. V okviru dela, ki ga definirajo opravila, sprejemajo poslovne odločitve, ki vplivajo na potek poslovnega procesa. Ker so deležniki opravila po navadi točno določeni vnaprej, želimo imeti mehanizem za obveščanje o rednih in izrednih poslovnih dogodkih tudi za drugo skupino ljudi. To skupino imenujemo prejemniki obvestil. V ta namen specifikacija predpisuje poseben tip opravila, t. i. obvestilo.

5.6.1 Kako razlikujemo med opravilom in obvestilom

Opravilo je definirano kot skupek akcij, ki jih mora uporabnik uspešno izvesti v nekem poslovnem procesu. Rezultat izvedbe opravila vpliva na nadaljevanje poslovnega procesa. Dokler se opravilo ne zaključi, se poslovni proces ne more nadaljevati.

Bistvena razlika pri obvestilu je, da interakcija med uporabnikom in obvestilom ne vpliva na nadaljevanje izvajanja poslovnega procesa. Iniciator

obvestila oziroma okolje, v katerem se je obvestilo kreiralo, ne čaka na izvedbo obvestila in nanjo tudi ne vpliva. Na primer če se zaključi okolje, v katerem se je prožilo obvestilo, se pri tem obvestilo ne zaključi.

5.6.2 Kaj lahko proži obvestilo in kako

Obvestilo se uporablja za obveščanje deležnikov ali skupine deležnikov poslovnega procesa o pomembnih poslovnih dogodkih. Na primer: opravilo je prekoračilo čas izvedbe, naročilo je potrjeno, zahteva je rešena. Obvestilo lahko tako kot opravilo kreira izvedljivi poslovni proces. Na primer obvestilo vodji, da se je določena faza poslovnega procesa zaključila. Iniciator obvestila pa je lahko tudi opravilo samo. Na primer ob odobritvi naročila stranke se pošlje obvestilo nabavni službi.

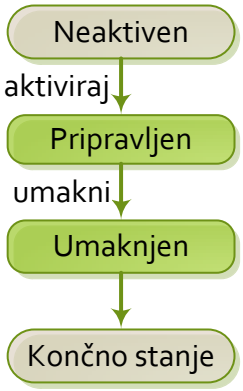
Primerjava lastnosti obvestil z opravilom:

1. lahko imajo več različnih naslovnikov in enega ali več poslovnih administratorjev;
2. generične vloge pri obvestilih ne igrajo nobene vloge;
3. obvestila tako kot opravilo vsebujejo elemente, ki podpirajo možnost vizualizacije poslovnih podatkov;
4. ne vplivajo na nadaljevanje procesa, v katerem so nastala;
5. iniciator obvestil nima vpliva na njihovo izvajanje;
6. ne podpirajo komentarjev, sprotnega dodajanja prilog in niso časovno omejena.

5.6.3 Življenjski cikel obvestil

Življenjski cikel obvestil je prikazan na sliki 5.3. Obvestila po aktivaciji preidejo v stanje *pripravljen* (angl. ready). V tem stanju so že na voljo za opazovanje in jih lahko s poizvedbo v obliki seznama pridobimo preko programskega vmesnika za podporo odjemalcem. V končno stanje *umaknjen*

(angl. removed) preidejo, ko uporabnik obvestilo odstrani iz seznama obvestil in s tem se življenjski cikel obvestila zaključi.



Slika 5.3: Življenjski cikel obvestila [1]

5.6.4 Časovne omejitve na opravilu

Največkrat se v poslovnem procesu pojavi zahteva, da moramo opravilom definirati časovne intervale, v okviru katerih morajo biti izvedene določene aktivnosti. Primer take zahteve je, ko opravilo časovno prekorači predviden rok izvedbe in želimo o tem obvestiti deležnike opravila, na katere ta poslovni dogodek vpliva.

V ta namen je predviden element <*htd:deadlines*>. Vsebuje dva gnezdena elementa, ki določata časovno omejitev. Rok za začetek opravila je definiran z elementom <*htd:startDeadline*> in rok za konec opravila z elementom <*htd:completionDeadline*>. Na opravilu lahko definiramo poljubno število časovnih omejitev. Struktura za definiranje časovnih omejitev je prikazana v naslednjem primeru kode 5.1.

Koda 5.1: Nastavitev časovnih omejitev na opravilu

```
<htd:task name="urejanjeZahteve">
  <htd:deadlines>
```

```
    <htd:startDeadline name="niPravocasnoZacel">
      ( <htd:for expressionLanguage="anyURI"? />
      | <htd:until expressionLanguage="anyURI"? /> )
      <htd:escalation name="obvestiVodjo">  ...  </htd:escalation>
    </htd:startDeadline>
    <htd:completionDeadline name="niPravocasnoKoncal">  ...
    </htd:completionDeadline>
  </htd:deadlines>
</htd:task>
```

Čas za začetek opravila se meri od prehoda opravila iz stanja *kreiran* v stanje *v teku*. Pravočasen začetek opravila izključi tudi časovnik. Podobno se meri čas za konec opravila, od prehoda opravila iz stanja *kreiran* do enega od končnih stanj opravila (*zaključen*, *neuspešno zaključen*, *v napaki*, *izhod*, *zastarel*). Ko opravilo doseže končno stanje, se izbrišejo vse njegove časovne omejitve.

Dolžino časovnih intervalov določamo z elementoma [23]:

- <*htd:for*> - vrednost je tipa "xsd:duration"in določa časovni interval, izražen z enoto časa (npr. P1Y5D pomeni 1 leto in 5 dni).

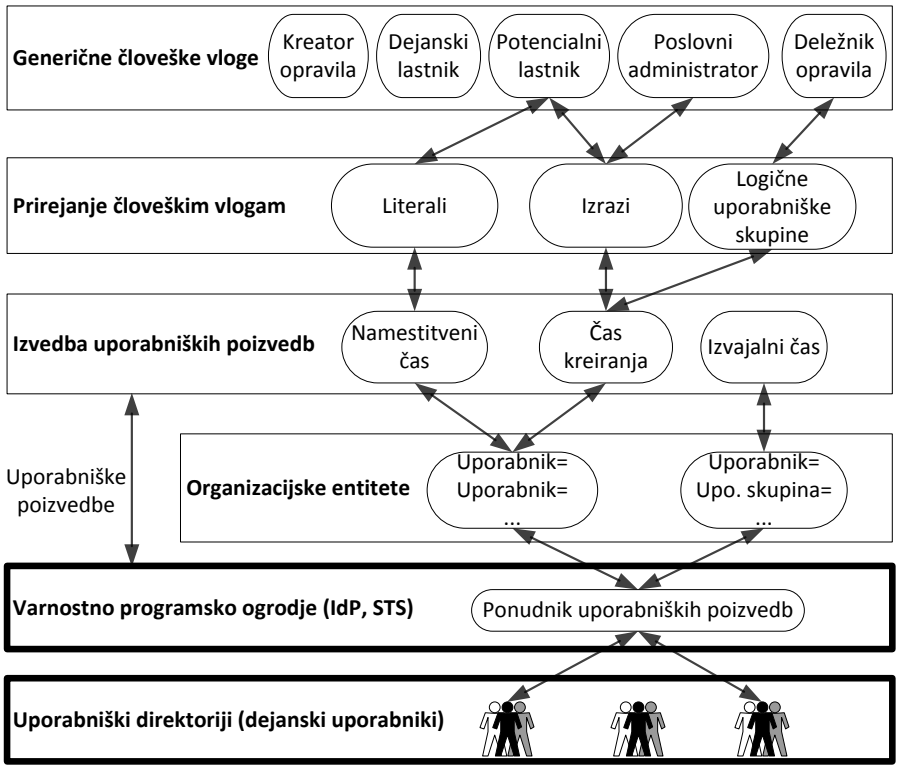
- <*htd:until*> - vrednost je tipa "xsd:DateTime"in določa absolutni čas (npr. 2014-05-11T10:00:00).

5.7 Varnostni vidiki

5.7.1 Varnostno programsko ogrodje

Pri prirejanju človeških vlog dejanskim uporabnikom je ena od pomanjkljivosti specifikacije BPEL4People varnostna opredelitev. Kako naj se izvajajo uporabniške poizvedbe in prirejajo organizacijske strukture človeškim vlogam, je prepuščeno zalednim sistemom. V tem podpoglavju bomo povzeli članek [11] na temo varnostnega programskega ogrodja in predstavili glavne varnostne koncepte pri prirejanju človeških vlog dejanskim uporabnikom.

Varnostno programsko ogrodje je zasnovano kot samostojna programska komponenta, ločena od upravitelja opravil. Avtentikacijo vseh akterjev in avtorizacijo vseh dostopov naj bi izvajala izključno ta komponenta po principu uporabe žetonov za dostop. Slika 5.4 prikazuje umestitev te komponente v sistem človeških poizvedb. Črno obrobljena pravokotnika predstavljata varnostni del, ki ni predmet specifikacije BPEL4People. Ostale nivoje, prikazane na sliki, smo že spoznali v podpoglavju 4.4.



Slika 5.4: Umestitev varnostnega programskega ogrodja v sistem človeških poizvedb [11]

Uporabnik, ki želi biti akter v konkretnem opravilu, mora najprej z zahtevo pridobiti varnostni žeton. Zahtevo prejme sistem za izdajo in validacijo

varnostnih žetonov STS (angl. Security Token Service), ki je del varnostnega programskega ogrodja. Sistem STS temelji na standardu WS-Trust [21]. Prijavne podatke obdela ponudnik identitet (angl. IdP), ki uporabi internetni protokol LDAP za dostop do uporabniških imenikov in tako preveri njihovo veljavnost.

Pridobljeni varnostni žeton uporabnik uporabi pri vsakem dostopu in je vezan na konkretno človeško opravilo. Na podlagi tega ga sistem lahko poveže z ustreznimi varnostnimi pravili.

5.7.2 Varnost s stališča delitve dolžnosti

V prejšnjih poglavjih smo že govorili o človeških vlogah in njihovi vlogi pri delegiranju in posredovanju opravil (podpoglavje 5.4.1). Pri tem se moramo zavedati še določenih omejitev, ki jih specifikaciji BPEL4People in Human Tasks natančneje ne upoštevata [15].

1. Specifikacija predpisuje, da se ob posredovanju opravila drugi osebi izločeni lastniki implicitno umaknejo iz množice potencialnih lastnikov, vendar ne pojasnjuje, kako se to zagotovi. Potencialni lastnik, ki ima ustrezne pravice za posredovanje opravila, lahko posreduje opravilo neavtoriziranemu uporabniku. V določenih primerih bi bilo smiselno posredovanje opravil onemogočiti in tako preprečiti scenarije, ki bi kršili princip delitve dolžnosti.

2. Ena od težav je tudi v uporabi sistemske funkcije *getActualOwner()* (podpoglavje 5.8.5). Funkcija vrača dejanskega lastnika na opravilu. Če definiramo delitev dolžnosti med dvema opraviloma, ki se izvajata hkrati in sta medsebojno izključujoči, se lahko eno opravilo začne v točki, ko drugo opravilo še nima določenega dejanskega lastnika. V tem primeru funkcija vrne prazno množico in posledično tudi ni določena množica izločenih lastnikov, ki so ključ za izpolnitev delitve dolžnosti. Iz tega sledi, da sta pomembna tako dejanski lastnik kot tudi zgodovina lastnikov instance opravila. Predlagana rešitev tega problema je

uporaba dveh različnih metod za pridobivanje informacije o dejanskem lastniku opravila.

3. Poslovni administrator je najmočnejša vloga in lahko posega v vsa stanja opravil. Glede na stopnjo zaupanja uporabniku, ki vlogo izpolnjuje, je to z varnostnega stališča zelo kritično. Zaledni sistem, ki implementira specifikaciji BPEL4People in Human Tasks, bi moral poslovnemu administratorju vračati informacijo o morebitnih kršitvah delitve dolžnosti med postopki posredovanja, delegiranja in nominacije.

5.7.3 Avtorizacija metod

Vse metode, ki jih bomo natančneje opisali v poglavju 5.8, so avtorizirane po principu človeških vlog. Če osebi pripadajo ustrezne človeške vloge in ima le-ta v okviru svojih vlog dovoljenje za izvajanje metod nad opravilom ali obvestilom, se metoda izvede, drugače se prožijo izjeme (podpoglavje 5.8.8). Nekaj metod in njihovih odvisnosti od človeških vlog je prikazanih v tabeli 5.1.

Tabela 5.1: Avtorizacija metod s človeškimi vlogami [4]

Metoda	**Iniciator opravila**	**Deležnik opravila**	**Potencialni lastnik**	**Dejanski lastnik**	**Poslovni admin**	**Prejemniki obvestil**
Claim	-	?	+	/	?	/
Start	-	?			?	/
Stop	-	?			?	/
Suspend	?	+	?	?	+	/
Complete	-	?	/	+	?	/
Forward	?	+	?	+	+	/
Nominate	?	-	-	-	+	-
getMyTaskDetails	+	+	+	+	+	+
setGenericHumanRole	-	-	-	-	+	-

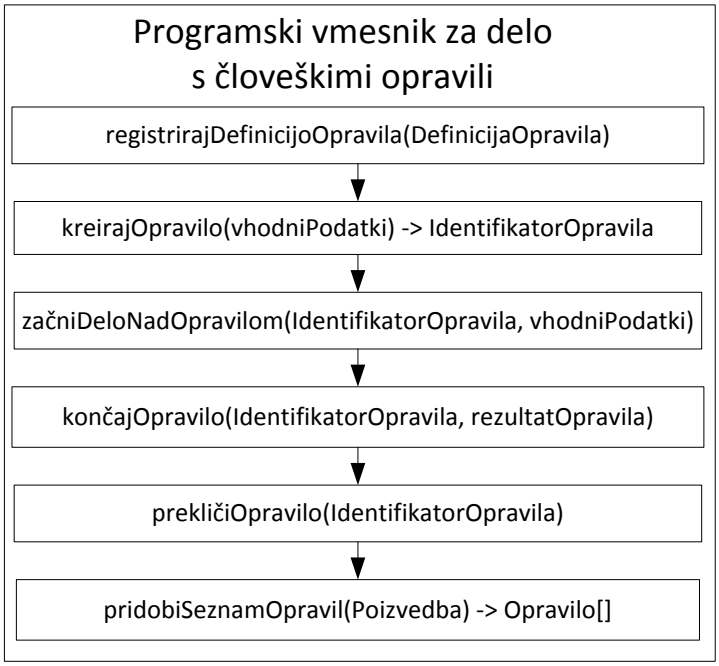
Znak "-" v tabeli 5.1 označuje, da vloga nima dovoljenja za metodo, in znak "+", da ima dovoljenje. Znak "/" označuje, da za opazovano vlogo metoda ne more biti izvedena. Znak "?" označuje, da je implementacija metode za konkretno vlogo prepuščena ponudniku.

5.8 Vmesnik API za podporo odjemalcem

V življenjski cikel opravila so vključeni tudi različni odjemalci. To so vse zunanje aplikacije, ki želijo nadzirati potek opravil in pridobiti njihove informacije. Zato mora strežniška infrastruktura za upravljanje z opravili omogočati dostop do informacij o stanju opravil. Obvezno mora zagotoviti predpisani programski vmesnik API, preko katerega lahko odjemalci izvajajo poizvedbe po opravilih, oziroma z njimi upravljajo. V nadaljevanju si bomo ogledali ključne metode za podporo odjemalcu in pri vsaki vrsti podrobneje navedli primer implementacije.

5.8.1 Komunikacija preko vmesnika API

Strežnik za izvajanje človeških opravil je implementiran tako, da ustreza arhitekturi SOA. Komunikacija z njim poteka preko spletnih storitev, ki jih predvideva specifikacija WSHumanTask. Zato je vmesnik API na strežniku odjemalcem dostopen preko spletne storitve. Diagram na sliki 5.5 prikazuje tipičen postopek uporabe vmesnika za delo s človeškimi opravili (angl. Human Tasks Server API) [20].



Slika 5.5: Osnovni način uporabe programskega vmesnika za podporo odjemalcem

Razvijalec definira človeško opravilo in definicijo registrira na strežnik. Če je opravilo definirano znotraj procesa BPEL, za njegovo registracijo poskrbi namestitev procesa BPEL. Definicija opravila vsebuje opis metode za kreiranje opravila in strukture vhodnega ter izhodnega sporočila. Pri registraciji strežnik preveri sintaktično pravilnost po predpisani shemi in vsebinsko pravilnost podatkov po pravilih, določenih v specifikaciji WSHumanTask.

Kasneje lahko opravilo kreirata proces ali uporabnik, neodvisno od procesa. Čeprav opravila največkrat povezujemo s procesi, je strežnik za izvajanje opravil neodvisen od procesov. Na primer uporabnik lahko kreira in upravlja z opravili neposredno preko upravitelja opravil.

Uporabnik ali zaledne aplikacije lahko seznam opravil pregledujejo, izvajajo poizvedbe po seznamu opravil in prevzemajo lastništvo nad opravili. Tako po uspešni kot po neuspešni izvedbi opravila uporabnik to sporoči s

klicem za zaključek ali preklic opravila.

5.8.2 Komunikacija odjemalca preko vmesnika API

Odjemalca lahko implementiramo v kakršnikoli tehnologiji (npr. JSP, PHP, .NET), le da podpira protokol SOAP za izmenjavo podatkov v formatu XML. Identiteta uporabnika, ki spletno storitev uporablja, se ugotovi preko avtorizacije samega klica storitve. Zapisana je v glavi sporočila SOAP.

Koda 5.2: Primer sporočila SOAP v primeru poizvedbe po seznamu opravil

```
POST https://localhost:9443/services/HumanTaskClientAPIAdmin/ HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: text/xml;charset=UTF-8
SOAPAction:
"http://docs.oasis-open.org/ns/bpel4people/ws-humantask/api/200803/
simpleQuery"
Authorization: Basic ZXZnZW46ZXZnZW5ldmdlbg==
Content-Length: 524
Host: localhost:9443
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/api/
  200803"
  xmlns:ns1="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/types/
  200803">
  <soapenv:Body>
    <ns:simpleQuery>
      <ns:simpleQueryInput>
        <ns1:simpleQueryCategory>ALL_TASKS</ns1:simpleQueryCategory>
      </ns:simpleQueryInput>
    </ns:simpleQuery>
  </soapenv:Body>
</soapenv:Envelope>
```

Koda 5.3: Primer sporočila SOAP, ki je odgovor strežnika na prejšnjo poizvedbo (koda 5.2)

```
HTTP/1.1 200 OK
Set-Cookie: JSESSIONID=162C0355F7C3D26D6BB6F8B9C783E518;
Path=/; Secure; HttpOnly
Content-Type: text/xml;charset=UTF-8
Transfer-Encoding: chunked
Date: Sat, 19 Apr 2014 16:03:11 GMT
Server: WSO2 Carbon Server
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv= "http://schemas.xmlsoap.org/soap/
envelope/">
  <soapenv:Body>
    <ns2:simpleQueryResponse xmlns:ns2=
    "http://docs.oasis-open.org/ns/bpel4people/ws-humantask/api/200803">
      <ns2:taskSimpleQueryResultSet>
        <row xmlns=
        "http://docs.oasis-open.org/ns/bpel4people/ws-humantask/types/ 200803">
          <id>4151</id>
          <taskType>TASK</taskType>
          <name xmlns:s9="http://diplomska/humantask">
            s9:ObdelavaZahteveTask
          </name>
          <presentationSubject>NazivZahteve</presentationSubject>
          <presentationName>Obdelava zahteve</presentationName>
          <status>READY</status>
          <priority>1</priority>
          <createdTime>2014-03-30T15:54:02.203+02:00</createdTime>
        </row>
        <ns1:pages xmlns:ns1=
        "http://docs.oasis-open.org/ns/bpel4people/ws-humantask/types/
        200803">1</ns1:pages>
      </ns2:taskSimpleQueryResultSet>
    </ns2:simpleQueryResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

5.8.3 Metode za spremembo stanja opravil

Uporabnik, ki ima dovolj pooblastil (npr. dejanski ali potencialni lastnik), s klicem metod za spremembo stanja opravil spreminja trenutno stanje opravil v sistemu. Pogoje, ki jim mora opravilo zadostiti za spremembo stanja, smo že opisali v podpoglavju 5.4.2. Če opravilo ni v pravem predhodnem stanju, metoda vrne izjemo.

Metoda **claim** spremeni trenutno odgovornost za opravilo. Uporabnik s klicem te metode prevzame lastništvo opravila in spremeni stanje opravila v *rezerviran*. Prehodno stanje opravila je *pripravljen*.

Tabela 5.2: Lastnosti metode claim

Vhodni parametri	**Izhodni parametri**	**Dovoljenje**	**Izjeme**
identifikator opravila	/	potencialni lastnik, poslovni administrator	IllegalStateFault, IllegalOperationFault, IllegalArgumentFault, IllegalAccessFault

5.8.4 Metode za administracijo

Administrativne metode tako kot ostale metode spreminjajo stanje opravila. Razlika med njimi je v nivoju uporabniškega dostopa. Za uporabo administrativnih metod mora uporabnik obvezno imeti vlogo poslovnega administratorja.

1. Metoda **activate** aktivira opravilo; premakne opravilo v stanje *pripravljen*;
2. Metoda **nominate** izvede nominacijo uporabnikov opravila;
3. Metoda **setGenericHumanRole** priredi drugo množico oseb eni od človeških vlog.

Metoda za nominacijo dodeli opravilu potencialne lastnike. Če je nominiran en uporabnik, opravilo preide v stanje *rezerviran*. Če je nominiranih

več oseb, opravilo preide v stanje *pripravljen*. Predhodno stanje opravila je *pripravljen*.

Tabela 5.3: Lastnosti metode nominate

Vhodni parametri	**Izhodni parametri**	**Dovoljenje**	**Izjeme**
identifikator opravila, organizacijska struktura	/	poslovni administrator	IllegalStateFault, IllegalOperationFault, IllegalArgumentFault, IllegalAccessFault

5.8.5 Razširjene sistemske metode jezika XPath

Razširjene sistemske metode jezika XPath uporabljamo pri definiciji človeških opravil. Omogočajo enostaven dostop do izvajalnega konteksta opravila v izvajalnem času in vračajo pomembne informacije o trenutnem stanju opravil. Zaledni sistem za upravljanje s človeškimi opravili mora glede na specifikacijo podpirati izvajanje teh metod. Zaradi združljivosti z jezikom XPath 1.0 funkcije ob napakah vračajo prazno množico.

1. **getPotentialOwners** vrača seznam potencialnih lastnikov opravila;
2. **getActualOwner** vrača dejanskega lastnika opravila;
3. **getTaskPriority** vrača prioriteto opravila;
4. **getInput** vrača informacijo o vhodnem sporočilu;
5. **getCountOfSubTasks** vrača število podrejenih opravil opazovanega opravila.

5.8.6 Enostavne metode za poizvedbo

Enostavne metode za poizvedbo po opravilih morajo biti obvezno del zalednega sistema za upravljanje s človeškimi opravili. V sistemu, glede na izbrane vhodne parametre metode, poiščejo ustrezna opravila in podrejena

opravila ter jih vrnejo v obliki seznama odjemalcu. Odjemalec lahko seznam uporabi v svoji aplikaciji za prikaz opravil.

1. **getMyTaskAbstracts** vrača seznam opravil in manjši nabor podatkov o opravilu;

2. **getMyTaskDetails** vrača seznam opravil in podrobnejši nabor podatkov o opravilu;

5.8.7 Napredne metode za poizvedbo

Napredne metode za poizvedbo po opravilih vračajo večji nabor informacij kot enostavne metode. Glede na specifikacijo jih zalednemu sistemu za upravljanje s človeškimi opravili ni treba implementirati, a so s stališča dostopa do podatkov opravil za odjemalce zaželena.

Predlagana je metoda **query**, ki deluje po principu strukturiranega poizvedovalnega jezika za delo s podatkovnimi bazami (angl. SQL).

Tabela 5.4: Lastnosti metode query

Vhodni parametri	**Izhodni parametri**	**Dovoljenje**	**Izjeme**
nabor želenih polj, pogojni stavek, sortiranje rezultata, maksimalno število opravil, odmik od maksimalnega števila opravil	tTaskQueryResultSet	ni omejitev	illegalStateFault, IllegalArgumentFault

Rezultat poizvedbe je vnaprej definirana struktura *tTaskQueryResultSet*. Struktura vsebuje nič ali več vrstic rezultata poizvedbe v vrstnem redu, ki je bil določen v vhodnem parametru metode. Rezultat je omejen s pogojnim stavkom in z maksimalnim številom opravil. Vrstica je definirana z elementom *tTaskQueryResultRow* in vsebuje množico podatkov o opravilu, med katerimi lahko izbiramo in jih določimo v vhodnem parametru metode.

5.8.8 Proženje izjem pri klicu metod programskega vmesnika

Metode za administracijo so namenjene uporabnikom, da z njimi neposredno vplivajo na stanje opravil v sistemu. Vsi vhodni parametri metod se vedno validirajo in so lahko v določenih primerih neustrezni. Enako velja za pogoje za spremembo stanja opazovanega opravila, ki niso vedno izpolnjeni. V obeh primerih se prožijo izjeme (angl. exceptions). Za ustrezno delovanje sistema mora krmilnik izjem pravilno obravnavati vse izjeme. Specifikacija zato določa naslednje tipe izjem:

- Pri izvedbi metode nad opravilom, ki je v nedovoljenem stanju, metoda proži izjemo *illegalStateFault*.
- Metode natančno opredeljujejo število in tip vhodnih parametrov. Kršitev povzroči izjemo *illegalArgumentFault*.
- Neavtoriziran uporabnik pri klicu metode nad opravili sproži izjemo *illegalAccessFault*. Ob obvestilu sproži izjemo tipa *recipientNotAllowed*.
- Izvedba metode nad napačnim tipom opravila proži izjemo *illegalOperationFault*.

Poglavje 6

Izdelava praktičnega primera

Za praktičen primer smo v jeziku BPEL opisali obstoječi poslovni proces sporočanja uporabniških zahtev v programersko podjetje (angl. bug report). Pod zahteve uvrščamo programske napake, težave uporabnikov pri delu s programom, povpraševanja po novih rešitvah itd. Obstoječi sistem je temeljil na namizni aplikaciji za vodenje internih zahtevkov. Stranka je svoje zahteve v podjetje sporočila preko elektronske pošte. Elektronska pošta se je ročno prenesla v program kot nova prispela zahteva. Podatki o pošiljatelju, vsebina elektronskega sporočila ter priponke so bili osnova za kreiranje uporabniške zahteve. Zahtevo se je glede na vsebino dodelilo odgovorni osebi, ki je določila izvajalca zahteve. Delo, ki ga je izvajalec opravil v okviru zahteve, je bilo ovrednoteno z urami, ki so bile osnova za obračun storitve in izdajo fakture stranki.

Pomanjkljivosti obstoječega sistema so naslednje:

- Ni omogočena elektronska oddaja zahtev in spremljanje oddanih zahtev v obliki seznama preko spleta.
- Elektronska sporočila se morajo ročno prenašati v sistem in ni časovnih omejitev, do kdaj je treba to storiti.
- Uporabnik ni samodejno obveščen o sprejeti zahtevi niti o stanju zahtev.

- Zahtevo se lahko prosto dodeljuje različnim osebam brez odgovornosti in ustreznega nadzora.

- Zahteve ostajajo neobdelane v sistemu dalj časa, časovnih omejitev pri tem ni.

Največja pomanjkljivost sistema s stališča strank je v pregledu oddanih zahtev in spremljanju, kaj se z zahtevami dogaja. Veliko zahtev ostaja nerešenih oziroma zanje ne prejmejo povratne informacije. S stališča podjetja ni sistemskega nadzora nad dolžino reševanja zahtev in odgovornostjo za zahteve.

6.1 Razvojno okolje in tehnologije

6.1.1 WSO2

Za strežniško infrastrukturo smo uporabili odprtokodno rešitev podjetja WSO2 [33]. Podjetje sestavljajo strokovnjaki z različnih področij, ki so sodelovali tudi pri definiranju standardov spletnih tehnologij (npr. XML, SOAP, Apache Axis/ Axis2, WSDL), ki so danes široko v uporabi. Vse njihove rešitve so odprtokodne in distribuirane pod licenco Apache Software License Version 2.0. Uporabili smo strežnik *WSO2 Business Process Server verzije 3.2.0*, ki podpira izvajanje izvedljivih poslovnih procesov na osnovi standarda WS-BPEL. Podpira tudi človeška opravila in notifikacije na osnovi specifikacij WS-BPEL 2.0 in BPEL4WS 1.1. Strežnik WSO2 vsebuje vgrajeni spletni pregledovalnik človeških opravil, v katerem avtorizirani uporabnik lahko upravlja s svojimi opravili. Na voljo je tudi grafični urejevalnik WSO2 Developer Studio, s katerim je mogoče enostavno in grafično modelirati poslovne procese. Za izvajanje poslovnih procesov, napisanih po standardu WS-BPEL, izkorišča izvajalno okolje Apache ODE (angl. Orchestration Director Engine) [2].

6.1.2 Intalio

Intalio [18] je komercialna in odprtokodna rešitev za upravljanje poslovnih procesov. Vsebuje tri komponente: strežnik Intalio BPMS server na arhitekturi Apache ODE z implementacijo jezika BPEL 2.0, orodje za načrtovanje Intalio Designer in Tempo WS-HumanTask service za upravljanje s človeškimi opravili. Načrtovanje poteka v notaciji BPM, ki se pri izvajanju na strežniku prevaja v jezik BPEL. Omogoča tudi migracijo procesov BPEL drugih ponudnikov, kot je na primer Oracle BPEL Process Manager.

6.1.3 Oracle SOA Suite

V maju 2015 je Oracle [19] objavil novo različico komercialne rešitve Oracle's BPM/SOA Suite 12c, ki vsebuje različna orodja za načrtovanje, nameščanje in upravljanje procesnih aplikacij. Za izvajanje poslovnih procesov uporablja Oracle BPEL Process Manager Runtime, ki podpira jezika BPEL 1.0 in 2.0. SOA Suite podpira standarde XML, WSDL in spletne storitve, XSLT, XPATH, JMS, JCA in druge, saj mu je veliko do standardizacije in interoperabilnosti med sistemi.

Podpira tudi človeške delovne tokove na podlagi opravil, ki pa niso kompatibilna s specifikacijo WS-HumanTask. V svoji različici podpirajo vse koncepte opravil, tudi notifikacije in eskalacije. Za opravila skrbi komponenta Human Task Service Component, ki uporabniku omogoča dostop preko vmesnika API, oziroma je na voljo grafična vizualizacija in spremljanje opravil preko spletne aplikacije Oracle BPM Worklist. Razvojno orodje, ki se imenuje Oracle JDeveloper, zagotovlja popolno podporo celotnemu razvojnemu ciklu.

6.1.4 ActiveVOS

ActiveVOS [3] je komercialna in odprtokodna rešitev za upravljanje s poslovnimi sistemi. Trenutna verzija 9, objavljena maja 2013, omogoča modeliranje procesov v notaciji BPMN 2.0 in implementacijo v jeziku BPEL 2.0. Za pod-

poro človeškim opravilom v celoti implementira specifikaciji BPEL4People 1.0 in WS-HumanTask 1.0 - z izjemo primera integracije samostojnih opravil v proces BPEL (primer C na sliki 4.1). ActiveVOS WS-HumanTask Service je samostojna komponenta za upravljanje z opravili, obvestili in eskalacijami. Preko vmesnika API nudi podporo različnim odjemalcem in pri tem uporablja protokola SOAP in JSON.

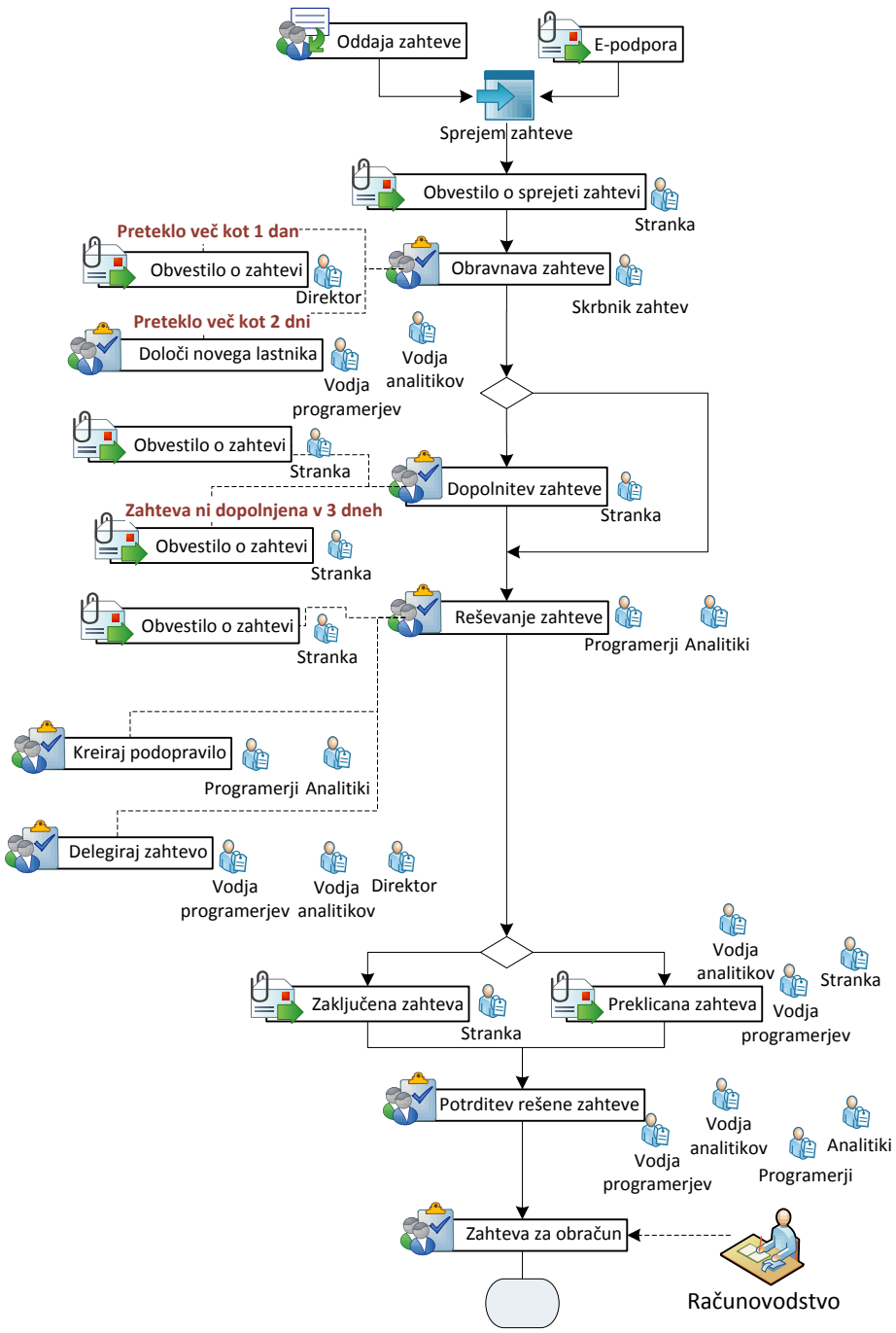
Dodatna strežniška infrastruktura ActiveVOS podpira standarde WS-*, WS-Policy, REST, Java Message Service (JMS), Enterprise Java Beans (EJB) in druge. Ponuja tudi dodatne funkcionalnosti, kot so modeliranje, simulacija, testiranje, razhroščevanje, spremljanje in poročanje. Strežnik ActiveVOS omogoča uporabo 32-bitnega in 64-bitnega izvajalnega okolja Java Virtual Machines (JVM) za operacijske sisteme Windows in Linux. Za načrtovanje, simulacijo in razvoj procesov je na voljo orodje ActiveVOS Designer na osnovni razvojnega orodja Eclipse.

6.1.5 OW2 Orchestra

Orchestra [24] je odprtokodna rešitev, napisana v programskem jeziku Java, in izdana pod licenco LGPL za podporo poslovnim procesom v jeziku BPEL. Temelji na konceptualnem modelu za podporo procesom PVM (angl. Process Virtual Machine) [29]. Zadnja različica Orchestra V.4 nudi podporo jeziku BPEL 2.0 in njegovi razširitvi BPEL4People ter implementira svojo komponento za podporo specifikaciji WS-HumanTask. Jedro rešitve temelji na odprtokodni knjižnici Apache CXF za razvoj storitev, ki nudi podporo standardom WS-Addressing in WS-ReliableMessaging. Za načrtovanje je mogoče uporabiti orodje Eclipse BPEL designer ali Netbeans BPEL designer. Za administracijo in spremljanje procesov je na voljo tudi spletna konzola oziroma spletni grafični vmesnik za načrtovanje, imenovan Web Designer BPMN 2.0.

6.2 Rešitev problema in opis procesa

Poslovni proces je grafično prikazan na sliki 6.1. Oddaja zahteve **stranke** v sistem proži začetek procesa. Zahtevo se lahko odda preko zunanjega sistema, in sicer spletne aplikacije ali elektronske pošte. Ker je poslovni proces na voljo kot spletna storitev, je mogoče oddajo zahteve integrirati tudi v druge programske rešitve.



Slika 6.1: Poslovni proces spremljanja zahtev stranke

Po uspešni oddaji zahteve stranka preko elektronske pošte prejme obvestilo o sprejeti zahtevi. Svoje zahteve in obvestila o poteku reševanja zahtev lahko spremlja preko spletne aplikacije za pregled zahtev. V naslednjem koraku procesa sistem zahteve obdela in jih na podlagi podanih informacij poskusi ustrezno kategorizirati (programska napaka, povpraševanje, ponudba, drugo).

Ker zahteva v splošnem ne vsebuje dovolj informacij za pričetek dela, mora oseba, ki jo imenujemo **skrbnik zahtev**, zahteve pregledati in urediti oziroma dopolniti in tako omogočiti nadaljevanje procesa. Ker mora biti odzivni čas s stališča stranke čim krajši, mora sistem zagotoviti, da so zahteve obravnavane pravočasno. Skrbnik zahtev je dolžan zahteve urediti najkasneje v enem dnevu, v nasprotnem primeru se o neurejeni zahtevi obvesti **direktorja**. Če zahteve vseeno še niso urejene po dveh dneh, jih samodejno prenesemo na drugo odgovorno osebo. Če skrbnik zahtev nima vseh potrebnih podatkov, lahko kreira opravilo stranki, da zahtevo dopolni.

Obravnavane zahteve so pripravljene na reševanje. Zahteve se v obliki človeških opravil dodelijo ustrezni skupini zaposlenih, odvisno od problemske domene zahteve. V našem primeru organizacijska struktura podjetja deli zaposlene na **analitike** in **programerje**. Vsaka skupina ima določenega vodjo skupine. Poimenovali smo ju **vodja analitikov** in **vodja programerjev**. Njuna vloga je dodeljevanje zahtev v reševanje svojim podrejenim in prevzemanje odgovornosti ob težavah pri reševanju zahtev.

Zahteve potrebujejo za izvedbo izvajalca. Izvajalec je vsak zaposleni, ki ima potrebno vsebinsko znanje za reševanje zahteve. Zahtevo lahko izvajalcem delegira odgovorna oseba oziroma v našem primeru vodja skupine. Izvajalec s prevzemom opravila postane dejanski lastnik opravila in tako lahko prične z delom na zahtevi. Druga možnost je, da odgovorna oseba neposredno kreira podopravilo izvajalcu. O prevzemu zahteve v reševanje je obveščena tudi stranka.

Poslovni rezultat procesa so seveda rešene zahteve stranke. V našem procesu so lahko zahteve zaključene ali preklicane. V obeh primerih je o tem

obveščena tudi stranka. Da se izognemo morebitnim konfliktom s stranko, morata število ur, porabljenih v okviru zahteve, potrditi tako odgovorna oseba kot izvajalec.

Zaključene zahteve morajo tudi skozi proces za obračun storitev, za kar poskrbi zunanji **računovodski sistem**, ki na koncu za opravljene storitve izda fakturo stranki. Računovodski sistem v našem procesu nastopa kot zunanji sistem. Omogočene so mu poizvedbe preko programskega vmesnika za dostop do opravil in pregled zahtev v obliki seznama, na podlagi katerega se izda faktura stranki. Ta del je neodvisen od našega izvedljivega procesa.

6.3 Načrtovanje in implementacija

6.3.1 Deležniki procesa

Iz opisa procesa smo opredelili naslednje deležnike in njihove vloge:

- **Stranka** je iniciator procesa. Njena vloga je oddaja in spremljanje zahtev.
- **Skrbnik zahtev** je odgovorna oseba za obravnavno vhodnih zahtev in zbiranje informacij za nadaljnjo obdelavo.
- **Analitiki** so zaposleni, ki imajo znanje za reševanje zahtev, ki vsebujejo analitične probleme.
- **Vodja analitikov** je oseba, odgovorna za skupino analitikov. Hkrati nastopa tudi v vlogi analitika.
- **Programerji** so zaposleni, ki imajo znanje za reševanje zahtev, ki vsebujejo programerske probleme.
- **Vodja programerjev** je oseba, odgovorna za skupino programerjev. Hkrati nastopa tudi v vlogi programerja.
- **Referenti** so vsi zaposleni v podjetju, ki lahko opravljajo delo na zahtevah.

- **Direktor** je direktor podjetja in najvišja uporabniška vloga v organizacijski strukturi podjetja.

- **Računovodstvo** je uporabniška vloga na opravilih, ki jih obravnava računovodstvo.

6.3.2 Proces BPEL

1. **Zajem zahtev**
   Proces, izpostavljen kot spletna storitev, preko dveh metod, **OddajaZahteve** in **OddajaZahteveEmail**, sprejema dva različna tipa sporočil. Definirata vhodno zahtevo in se v shemi razlikujeta po količini vhodnih podatkov. Podatke vhodnega sporočila z uporabo aktivnosti *Assign* in transformacij XSL shranimo v globalno spremenljivko **internaZahteva**, ki skozi celoten proces vzdržuje podatke o vhodni zahtevi (primer kode 6.1).

Koda 6.1: Zajem vhodnih zahtev stranke

```
<bpel:pick name="VhodneZahteve" createInstance="yes">
  <bpel:onMessage
      partnerLink="ZahteveObdelavaPL"
      operation="OddajaZahteve"
      portType="tns:ZahteveObdelavaProcesPT"
      variable="inputVarZahteva">
    <bpel:sequence name="SequenceZahteva">
      <bpel:assign name="AssignZahteva">
        <bpel:copy>
          <bpel:from>
            bpel:doXslTransform("ZahtevaProces.xsl",
            $inputVarZahteva.ZahteveObdelavaProcesMessage)
          </bpel:from>
          <bpel:to variable="internaZahteva"></bpel:to>
        </bpel:copy>
      </bpel:assign>
    </bpel:sequence>
  </bpel:onMessage>
  <bpel:onMessage
```

```
      partnerLink="ZahteveObdelavaPL"
      operation="OddajaZahteveEmail"
      portType="tns:ZahteveObdelavaProcesPT"
      variable="inputVarZahtevaEmail">
    <bpel:sequence name="SequenceZahtevaEmail">
      <bpel:assign name="AssignZahtevaEmail">
        <bpel:copy>
          <bpel:from>
            bpel:doXslTransform("ZahtevaEmail.xsl",
            $inputVarZahtevaEmail.ZahtevaObdelavaProcesEmailMessage)
          </bpel:from>
          <bpel:to variable="internaZahteva"></bpel:to>
        </bpel:copy>
      </bpel:assign>
    </bpel:sequence>
  </bpel:onMessage>
</bpel:pick>
```

2. **Obravnava zahteve in dopolnitev**

   Zajeto zahtevo smo prej transformirali v spremenljivko *internaZahteva*. V nadaljevanju procesa se pokliče opravilo za obravnavo zahteve. Referent pri obdelavi zahteve izpolni vse potrebne podatke za nadaljnjo obdelavo zahteve. Če zahteva ni ustrezna, to na zahtevi označi s statusom *V_DOPOLNITVI*. Stranka je o dopolnitvi obveščena z obvestilom in kreira se ji opravilo za dopolnitev zahteve.

Koda 6.2: Obravnava zahteve in dopolnitev

```
<-- obdelava zahteve ->
<bpel:extensionActivity>
  <b4p:peopleActivity name="SprejetaZahtevaObdelajTask"
  inputVariable="inputObdelavaZahteve"
  outputVariable="outputObdelavaZahteve">
    <b4p:remoteTask partnerLink="ObdelavaZahtevePartnerLink"
    operation="obdelava" responseOperation="obdelavaResponse">
    </b4p:remoteTask>
  </b4p:peopleActivity>
</bpel:extensionActivity>
<-- pogoj za dopolnitev zahteve ->
```

```
<bpel:if name="IfStatusZahteve">
  <bpel:condition>
    <![CDATA[
    $outputObdelavaZahteve.ObdelavaZahteveResponse/zots:statusZahteve =
    'V_DOPOLNITVI']]>
  </bpel:condition>
  <bpel:sequence name="Dopolnitev">
    ...
    <-- obvestilo o dopolnitvi zahteve ->
    <bpel:extensionActivity>
      <b4p:peopleActivity name="DopolnitevZahteveNotification"
      inputVariable="inputDopolnitevZahteve"
      outputVariable="outputDopolnitevZahteve">
        <b4p:remoteNotification
        partnerLink="ObvestiloOZahteviNotificationPartnerLink"
        operation="obvesti">
        </b4p:remoteNotification>
      </b4p:peopleActivity>
    </bpel:extensionActivity>
    ...
    <-- dopolnitev zahteve ->
    <bpel:extensionActivity>
      <b4p:peopleActivity name="DopolnitevZahteveTask"
      inputVariable="inputDopolnitevZahteve"
      outputVariable="outputDopolnitevZahteve">
        <b4p:remoteTask partnerLink="DopolnitevZahtevePartnerLink"
        operation="dopolnitev" responseOperation="dopolnitevResponse">
        </b4p:remoteTask>
      </b4p:peopleActivity>
    </bpel:extensionActivity>
  </bpel:sequence>
</bpel:if>
```

3. **Reševanje zahteve**

   Edini prvi pogoj za reševanje zahteve je ustrezno dopolnjena zahteva. Ta pogoj je izpolnjen že v postopku obravnave zahteve, zato poslovni proces pred kreiranjem zahteve za reševanje ne zahteva dodatne poslovne logike. Pred kreiranjem opravila za reševanje zahteve se pošlje stranki obvestilo, da je bila zahteva poslana v reševanje.

Potencialni lastnik je v med kreiranjem opravila točno določen, če je bila med obravnavo zahteve določena odgovorna oseba, v nasprotnem primeru so potencialni lastniki določeni glede na vsebino in področje zahteve (analitični ali programerski problem).

Rezultat rešene zahteve shranimo v spremenljivko *internaZahteva*. O rešeni zahtevi se obvesti stranko. Če je bila zahteva v času reševanja preklicana, je o tem obveščen tudi vodja.

Koda 6.3: Nastavitev potencialnih lastnikov opravila za reševanje zahtev

```
<bpel:if name="IfOdgovornaOseba">
  <bpel:condition>
    <![CDATA[
      count(
      $internaZahteva/zahtevaInterno:OdgovornaOseba/htt:organizationalEntity
      ) > 0
    ]]>
  </bpel:condition>
  <bpel:assign name="AssignOdgovornaOseba" >
    <from>
      $internaZahteva/zahtevaInterno:OdgovornaOseba/htt:organizationalEntity
    </from>
    <to>$zaposleni</to>
  </bpel:assign>
  <bpel:elseif>
    <bpel:condition>
      <![CDATA[
        $internaZahteva/zahtevaInterno:PodrocjeZahteve = 'ANALITIK'
      ]]>
    </bpel:condition>
    <bpel:assign name="AssignAnalitik" >
      <from>b4p:getLogicalPeopleGroup("Internal/Analitik")</from>
      <to>$zaposleni</to>
    </bpel:assign>
  </bpel:elseif>
  <bpel:elseif>
    <bpel:condition>
      <![CDATA[
```

```
        $internaZahteva/zahtevaInterno:PodrocjeZahteve = 'PROGRAMER'
      ]]>
    </bpel:condition>
    <bpel:assign name="AssignProgramer" >
      <from>b4p:getLogicalPeopleGroup("Internal/Programer")</from>
      <to>$zaposleni</to>
    </bpel:assign>
  </bpel:elseif>
  <bpel:else>
    <bpel:assign name="AssignDefault" >
      <from>b4p:getLogicalPeopleGroup("Internal/Referenti")</from>
      <to>$zaposleni</to>
    </bpel:assign>
  </bpel:else>
</bpel:if>
...
<bpel:extensionActivity>
  <b4p:peopleActivity name="ResevanjeZahteveTask">
    <b4p:remoteTask partnerLink="ResevanjeZahtevePartnerLink"
    operation="resevanje" responseOperation="resevanjeResponse">
      <htd:peopleAssignments>
        <htd:potentialOwners>
          <htd:from>$zaposleni</htd:from>
        </htd:potentialOwners>
      </htd:peopleAssignments>
    </b4p:remoteTask>
  </b4p:peopleActivity>
</bpel:extensionActivity>
```

4. **Potrditev rešene zahteve**

   Potrjevanje rešene zahteve poteka po principu delitve dolžnosti. Vsak izvajalec na zahtevi mora svoje delo potrditi kot zaključeno in vnesti dejansko število ur za obračun. V procesu se v zanki kreira eno opravilo za vsakega izvajalca in vsako odgovorno osebo na zahtevi (primer kode 6.4). Potencialni lastniki na opravilu so torej vsi deležniki zahteve. Izločeni lastniki so osebe, ki so zahtevo že potrdili. Ko so potrjena vsa opravila vseh deležnikov zahteve, se proces nadaljuje.

Koda 6.4: kreiranje opravil izvajalcem za potrditev rešene zahteve

```
<bpws:forEach parallel="no" counterName="Izvajalec" name="ForEach">
<bpws:startCounterValue>
  <![CDATA[1]]>
</bpws:startCounterValue>
<bpws:finalCounterValue>
  <![CDATA[
    count ( distinct-values (
      $internaZahteva/zahtevaInterno:OpravljenoDelo/OsebaDelo
    ))
  ]]>
</bpws:finalCounterValue>
<bpws:scope>
  <bpws:sequence>
    ...
    <extensionActivity>
      <b4p:peopleActivity name="PotrditevZahteveTask"
      inputVariable="inputPotrditevZahteve"
      outputVariable="outputPotrditevZahteve">
        <b4p:remoteTask partnerLink="PotrditevZahtevePartnerLink"
        operation="potrditevZahteve"
        responseOperation="potrditevResponse">
        </b4p:remoteTask>
      </b4p:peopleActivity>
    </extensionActivity>
    <!-- shranimo osebo, ki je že potrdila zahtevo -->
    <assign name="assignIzvajalecPotrdil">
      <copy>
        <from>
          <![CDATA[
          $outputPotrditevZahteve.PotrditevZahteveResponse/zots:OsebaPotrdil
          ]]>
        </from>
        <to><![CDATA[$IzvajalecPotrdil/izvajalec[Izvajalec]]]></to>
      </copy>
    </assign>
  </bpws:sequence>
</bpws:scope>
</bpws:forEach>
```

5. **Obračun zahteve**

   Obračun zahteve je s stališča procesa najbolj enostavno opravilo, saj ne potrebuje nobene obdelave rezultata opravila. Ko zunanji računovodski sistem zaključi opravilo, se zaključi tudi celoten proces. Pogoja za kreiranje opravila sta ustrezna vrsta dela (programer, analitik) in skupno število ur dela na zahtevi, ki mora biti večje od nič.

Koda 6.5: Obračun zahteve

```
<bpel:if name="IfDeloOpravljeno">
<bpel:condition>
  <![CDATA[
    count(
      $internaZahteva/zahtevaInterno:OpravljenoDelo[
      (./zahtevaInterno:VrstaDela = 'PROGRAMER' or
      ./zahtevaInterno:VrstaDela = 'ANALITIK') and
      ./zahtevaInterno:Kolicina > 0 ]
    ) > 0
  ]]>
</bpel:condition>
<bpel:extensionActivity>
  <b4p:peopleActivity name="ObracunZahteveTask"
  inputVariable="inputObracunZahteve"
  outputVariable="outputObracunZahteve">
    <b4p:remoteTask partnerLink="ObracunZahtevePartnerLink"
    operation="obracunZahteve"
    responseOperation="obracunResponse"></b4p:remoteTask>
  </b4p:peopleActivity>
</bpel:extensionActivity>
</bpel:if>
```

6.3.3 Uporabljena človeška opravila

1. **Obravnava vhodne zahteve**

   Opravilo je namenjeno obravnavi vhodne zahteve. Potencialni lastnik opravila mora pravočasno opremiti opravilo z informacijami, potrebnimi za uspešno nadaljevanje procesa. Na zahtevi uredi oziroma do-

polni vsebino, opredeli področje zahteve (analitik, programer, neopredeljeno) in izbere vrsto zahteve (zahteva, povpraševanje). Glede na vsebinsko področje zahteve lahko določi tudi odgovorno osebo. V okviru zahteve se vodi tudi interni status, ki se prednastavi na *Poslana zahteva*. Lastnosti opravila so:

- Vhodno in izhodno sporočilo opravila je definirano s strukturo po shemi **ZahtevaInterno.xsd**.
- Potencialni lastnik zahteve je določen s človeško vlogo *Skrbnik zahteve*. Delegiranje zahteve ni dovoljeno.
- Iniciator opravila je uporabnik, ki je oddal zahtevo.
- Poslovni administrator je določen s človeško vlogo *Direktor*.
- Opravilo je zaključeno, ko to referent potrdi s klikom na gumb *Potrdi zahtevo* oziroma odda zahtevo za dopolnitev z gumbom *Pošlji v dopolnitev*.
- V okviru opravila sta določeni dve eskalaciji (primer kode 6.7).
  (a) Po preteku enega dneva od kreiranja opravila se pošlje obvestilo *direktorju*.
  (b) Po preteku dveh dni od kreiranja opravila se opravilo dodeli skupini *referentov*. Ti postanejo novi potencialni lastniki opravila.
- Definiran je uporabniški vmesnik za urejanje podatkov o zahtevi (glej podpoglavje 6.4).

Koda 6.6: Deležniki opravila za obravnavo vhodne zahteve

```
<htd:peopleAssignments>
  <!-- zahteva se dodeli skrbnikom zahtev -->
  <htd:potentialOwners>
    <htd:from logicalPeopleGroup="Internal/SkrbnikiZahtev">
      <htd:argument name="role">
        Internal/SkrbnikiZahtev
      </htd:argument>
```

```
    </htd:from>
  </htd:potentialOwners>
  <!-- iniciator zahteve je uporabnik, ki je oddal zahtevo -->
  <htd:taskInitiator>
    <htd:from logicalPeopleGroup="Internal/Stranka">
      <htd:argument name="person">
        htd:getInput("ObdelavaZahteveRequest")/zots:OsebaNarocnik
      </htd:argument>
    </htd:from>
  </htd:taskInitiator>
  <!-- poslovni administrator je direktor -->
  <htd:businessAdministrators>
    <htd:from logicalPeopleGroup="Internal/Direktorji">
      <htd:argument name="role">
        Internal/Direktorji
      </htd:argument>
    </htd:from>
  </htd:businessAdministrators>
</htd:peopleAssignments>
<htd:delegation potentialDelegatees="nobody" />
```

Koda 6.7: Eskalacija in delegiranje zahteve

```
<!-- opravilu dodamo časovne alarme -->
<htd:deadlines>
  <htd:startDeadline name="zamudilPrvic">
    <!-- če se zahteva ni začela obravnavati v 1 dnevu,
                              obvestimo direktorja -->
    <htd:for>P1D</htd:for>
    <htd:escalation name="obvestiloDirektorju">
      <htd:toParts>
        <htd:toPart name="VsebinaObvestila">
          htd:getInput("ObdelavaZahteveRequest")/zots:VsebinaZahteve
        </htd:toPart>
        <htd:toPart name="NazivObvestila">
          htd:getInput("ObdelavaZahteveRequest")/zots:NazivZahteve
        </htd:toPart>
      </htd:toParts>
      <htd:localNotification reference="tns:ObvestiloOZahteviNotification">
        <htd:potentialOwners>
          <htd:from logicalPeopleGroup="Internal/Direktor">
```

```
            <htd:argument name="role">Internal/Direktor</htd:argument>
          </htd:from>
        </htd:potentialOwners>
      </htd:localNotification>
    </htd:escalation>
  </htd:startDeadline>
  <htd:startDeadline name="zamudilDrugic">
    <!-- če se zahteva ni začela obravnavati po 2 dneh,
         dodelimo referentom -->
    <htd:for>P2D</htd:for>
    <htd:escalation name="dodeliReferentom">
      <htd:reassignment>
        <htd:potentialOwners>
          <htd:from logicalPeopleGroup="Internal/Referenti">
            <htd:argument name="role">Internal/Referenti</htd:argument>
          </htd:from>
        </htd:potentialOwners>
      </htd:reassignment>
    </htd:escalation>
  </htd:startDeadline>
</htd:deadlines>
```

2. **Dopolnitev zahteve**

   Opravilo je namenjeno stranki za dopolnitev zahteve. Razlika z opravilom za obravnavo zahteve je, da je uporabniški vmesnik prirejen stranki in je mogoče urejati samo določene podatke. Lastnosti opravila so:

   - Vhodno in izhodno sporočilo opravila je definirano s strukturo po shemi **ZahtevaInterno.xsd**.
   - Potencialni lastnik opravila je uporabnik, ki je zahtevo oddal in se ga dinamično nastavi glede na podatke v vhodnem sporočilu o uporabniku, ki je zahtevo oddal. Delegiranje zahteve ni dovoljeno.
   - Iniciatorja opravila se dinamično nastavi glede na podatke v vhodnem sporočilu o uporabniku, ki je zahtevo oddal.
   - Poslovni administrator je določen s človeško vlogo *Direktor*.

- Opravilo je zaključeno, ko to uporabnik potrdi s klikom na gumb *Oddaj dopolnitev*.
- V okviru opravila je določena eskalacija. Stranka mora zahtevo dopolniti v 3 dneh, drugače dobi obvestilo.
- Uporabniški vmesnik prikazuje samo ključne podatke, ki jih potrebuje stranka za dopolnitev zahteve (glej podpoglavje 6.4).

3. **Reševanje zahteve**

   Reševanje zahteve je glavno človeško opravilo znotraj poslovnega procesa. Referent oziroma dejanski lastnik opravila lahko pregleduje in ureja vse podatke o vhodni zahtevi. Lahko vpiše količino ur, ki jih je potreboval za opravljeno delo ali kreira podopravila drugim potencialnim lastnikom.

   - Vhodno in izhodno sporočilo opravila je definirano s strukturo po shemi **ZahtevaInterno.xsd**.
   - Potencialni lastniki opravila so človeške vloge *Programerji* oziroma *Analitiki*, določeni glede na vrsto zahteve. Če vrste zahteve ni mogoče opredeliti, lahko potencialnega lastnika določa tudi človeška vloga *Referenti*, ki združuje vse referente. Delegiranje zahteve je dovoljeno med referenti in vodji.
   - Iniciatorja opravila se določi s poizvedbo po podatku v vhodnem sporočilu o uporabniku, ki je zahtevo oddal.
   - Poslovni administrator je določen s človeško vlogo *Direktor*.
   - Opravilo je zaključeno, ko to uporabnik potrdi s klikom na gumb *Zaključi zahtevo*, *Zavrni zahtevo* ali *Pošlji v dopolnitev*.
   - Definiran je uporabniški vmesnik za urejanje podatkov o zahtevi. Vsebuje vse informacije o zahtevi. Omogočeno je urejanje podatkov in dodajanje novega dela na zahtevo.

Koda 6.8: Definicija podopravila na opravilu za reševanje zahtev

```
<htd:tasks>
...
  <htd:composition type="parallel" instantiationPattern="manual">
    <htd:subtask name="DeloNaZahtevi">
      <htd:task>
       ... definicija opravila za Delo na zahtevi ...
      </htd:task>
     <htd:toParts>
       <htd:toPart name="ZahtevaInput">
         htd:getInput("ResevanjeZahteveRequest")
       </htd:toPart>
     </htd:toParts>
  </htd:composition>
  <htd:completionBehavior>
    <htd:completion>
      <htd:result>
        <htd:copy>
          <htd:from>
            htd:getSubtaskOutputs("DeloNaZahtevi",
            "deloNaZahteviResponse", "/OpravljenoDelo")
          </htd:from>
          <htd:to>/OpravljenoDelo</htd:to>
      </htd:result>
    </htd:completion>
  </htd:completionBehavior>
</htd:task>
```

4. **Potrditev rešene zahteve**

   Opravilo je primer delitve dolžnosti, kjer morata tako vodja skupine kot izvajalec opravila potrditi rešeno zahtevo. Za realizacijo delitve dolžnosti se kreirata dve isti opravili, opravilo A in B, pri čemer imata istega potencialnega lastnika. Vodja skupine je hkrati tudi član skupine. Lastnosti opravila so:

   - Vhodno in izhodno sporočilo opravila je definirano s strukturo po shemi **ZahtevaInterno.xsd**.

- Potencialna lastnika zahteve sta dve osebi, vodja skupine in izvajalec. Delegiranja zahteve ni dovoljeno.
- Izločeni lastnik zahteve A ni določen. Zahteva B ima izločenega lastnika, določenega z izrazom *b4p:getActualOwner("A")*. Vrstni red potrjevanja opravila v tem primeru ni pomemben.
- Iniciator opravila je referent, ki je zadnji reševal zahtevo.
- Poslovni administrator je določen s človeško vlogo *Direktor*.
- Opravilo je zaključeno, ko to uporabnik potrdi s klikom na gumb *Potrdi zaključeno zahtevo*.
- V okviru opravila ni določene eskalacije.
- Uporabniški vmesnik prikazuje podatke o zahtevi in opravljenem delu na zahtevi.

5. **Zahteva za obračun**

   Zahteva za obračun je človeško opravilo, namenjeno zunanjemu računovodskemu sistemu. Ta preko programskega vmesnika za podporo odjemalcem dostopa do seznama svojih nezaključenih zahtev, na podlagi katerega izvede obračun storitev. Lastnosti opravila so:

   - Vhodno in izhodno sporočilo opravila je definirano s strukturo po shemi **ZahtevaInterno.xsd**.
   - Potencialni lastnik zahteve je določen s človeško vlogo *Računovodstvo*. Delegiranje zahteve ni dovoljeno.
   - Iniciator opravila je vodja skupine, ki je reševala zahtevo.
   - Poslovni administrator je določen s človeško vlogo *Direktor*.
   - Opravilo je zaključeno, ko zunanji sistem zaključi opravilo z metodo *Complete(task identifier)*.
   - V okviru opravila ni določene eskalacije.

- Uporabniški vmesnik ni definiran. Z opravilom ni neposredne uporabniške komunikacije. Računovodski sistem pridobi vse informacije o opravilu preko programskega vmesnika za podporo odjemalcem.
- Pogoji za zaključek opravila so izpolnjeni, ko se spremeni status zahteve v *Obračunano*.

6.3.4 Uporabljene notifikacije

V našem primeru smo pri notifikacijah izkoristili možnost večkratne uporabe istega obvestila. Med izvajanjem procesa uporabnika sproti obveščamo o stanju zahteve in pri tem uporabljamo isto definicijo obvestila, imenovanega **ObvestiloOZahteviNotification**.

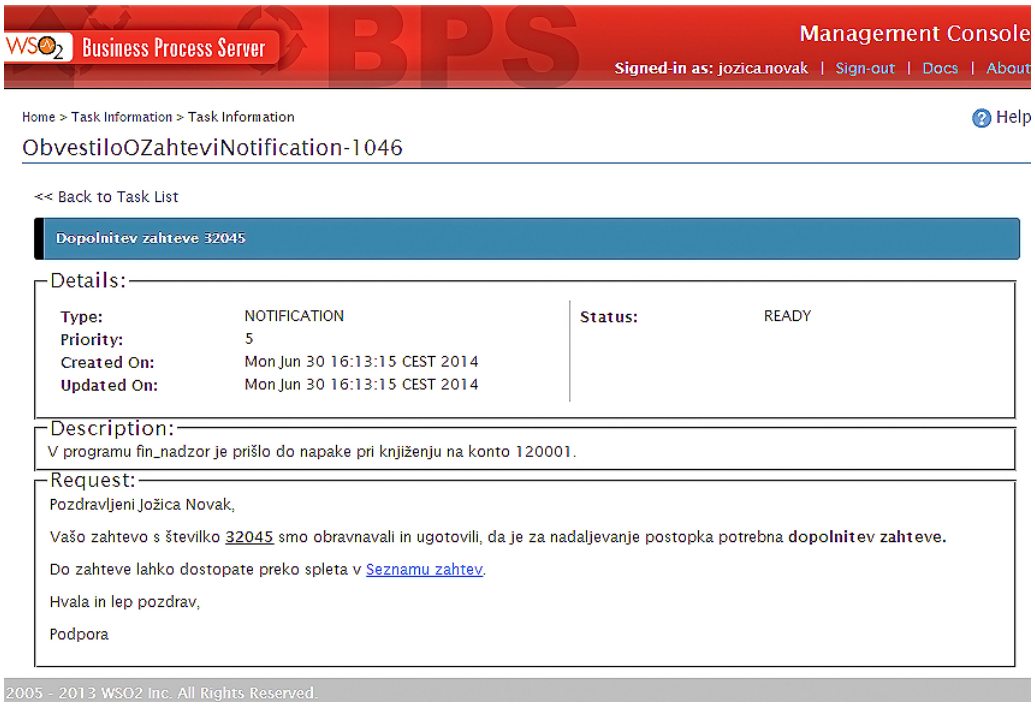
Sistem WSO2 omogoča tudi samodejno pošiljanje elektronskega sporočila ob kreiranju obvestila. Podatke za elektronsko sporočilo definiramo v okviru elementa *<htd:rendering type="wso2:email">* (primer kode 6.9).

- Vhodno sporočilo obvestila vsebuje prejemnika obvestila in podatke o zahtevi.
- Prejemnik sporočila se določi dinamično med izvajanjem. Prejemnik postane uporabnik, ki je v vhodnem sporočilu obvestila definiran kot prejemnik obvestila.
- Poslovni administrator je določen s človeško vlogo *Direktor*.
- Uporabniški vmesnik je enoten za vse variante obvestila. Prikazan je na sliki 6.2.

Koda 6.9: Definicija obvestila o zahtevi

```
<htd:notification name="ObvestiloOZahteviNotification">
<!-- uporabniška notifikacija - obvestilo o zahtevi -->
  <htd:interface portType="cl:ObvestiloOZahteviPLT" operation="obvesti" />
  <htd:peopleAssignments>
```

```
    <!-- prejemnik obvestila je definiran v vhodnem sporočilu -->
    <htd:recipients>
      <htd:from logicalPeopleGroup="Internal/Stranka">
        <htd:argument name="person">
          htd:getInput("ObvestiloOZahteviNotificationRequest")
          /zots:PrejemnikObvestila
        </htd:argument>
      </htd:from>
    </htd:recipients> ...
  </htd:peopleAssignments>
  <htd:presentationElements>
    <htd:presentationParameter name="NazivObvestila" type="xsd:string">
      htd:getInput("ObvestiloOZahteviNotificationRequest")/zots:NazivObvestila
    </htd:presentationParameter> ...
  </htd:presentationElements>
  <htd:renderings> ...
    <htd:rendering type="wso2:email"
    xmlns:wso2="http://wso2.org/ht/schema/renderings/">
    <!-- podatki za elektronsko sporočilo -->
      <wso2:to name="to" type="xsd:string">$PrejemnikObvestila$</wso2:to>
      <wso2:subject name="subject"
      type="xsd:string">$NazivObvestila$</wso2:subject>
      <wso2:body name="body"
      type="xsd:string">$VsebinaObvestila$</wso2:body>
    </htd:rendering>
  </htd:renderings>
</htd:notification>
```



Slika 6.2: Primer grafičnega prikaza obvestila o zahtevi v sistemu WSO2

6.3.5 Zunanji sistemi

V našem primeru je zunanji sistem računovodski program, napisan v programskem jeziku C#. Program preko API dostopa do opravil in nudi uporabniku seznam nezaključenih opravil, na podlagi katerih lahko izvede obračun storitev. V nadaljevanju bomo opisali postopek ter primer dostopa v programskem jeziku C# in uporabljene metode (koda 6.10).

Metode API so opisane z datoteko WSDL, ki je v sistemu WSO2 dostopna na spletnem naslovu [25]. Uporabnik se mora pred uporabo vmesnika najprej uspešno avtorizirati. Na podlagi pridobljenega varnostnega žetona mu pripada tudi določena človeška vloga oziroma pravice za upravljanje z opravili.

Koda 6.10: Odsek kode računovodskega programa za upravljanje z opravili v programskem jeziku C#

```
1  WSO2.taskOperationsClient client = new
   WSO2.taskOperationsClient("taskOperationsSOAP1");
2  client.ClientCredentials.HttpDigest.ClientCredential = new
   System.Net.NetworkCredential("janez", "janez");
3  client.Open();
4
5  WSO2.simpleQuery query1 = new WSO2.simpleQuery();
6  query1.simpleQueryInput = new WSO2.tSimpleQueryInput() {
7    simpleQueryCategory = WSO2.tSimpleQueryCategory.ALL_TASKS,
8    simpleQueryCategorySpecified = true,
9    taskName = "RacunovodstvoTask",
10   status = new string[]{"READY","RESERVED"}
11 };
12
13 WSO2.simpleQueryResponse response = client.simpleQuery(query1);
14 foreach (WSO2.tTaskSimpleQueryResultRow row in
   response.taskSimpleQueryResultSet.row) {
15   WSO2.loadTask load = new WSO2.loadTask();
16   load.identifier = row.id;
17   WSO2.loadTaskResponse result = client.loadTask(load);
18
19   WSO2.getInput input = new WSO2.getInput();
20   input.identifier = row.id;
21   WSO2.getInputResponse input_result = client.getInput(input);
22
23   //prenos in obdelava podatkov na opravilu
24   WSO2.complete complete = new WSO2.complete();
25   complete.identifier = row.id;
26   WSO2.completeResponse complete_resutl = client.complete(complete);
27 }
28 client.Close();
```

1. **Pridobitev seznama opravil in lastnosti**

   Seznam opravil smo pridobili z metodo za enostavno poizvedbo *simpleQuery*(). Vedno pridobimo seznam vseh opravil, ki so v statusu *pripravljeni* ali *rezervirani*. Dodatne informacije, ki jih vsebuje opravilo,

pridobimo s klicem metode *loadTask()*. Vhodno sporočilo opravila, ki vsebuje vse pomembne podatke za obdelavo, pridobimo s klicem metode *getInput()*.

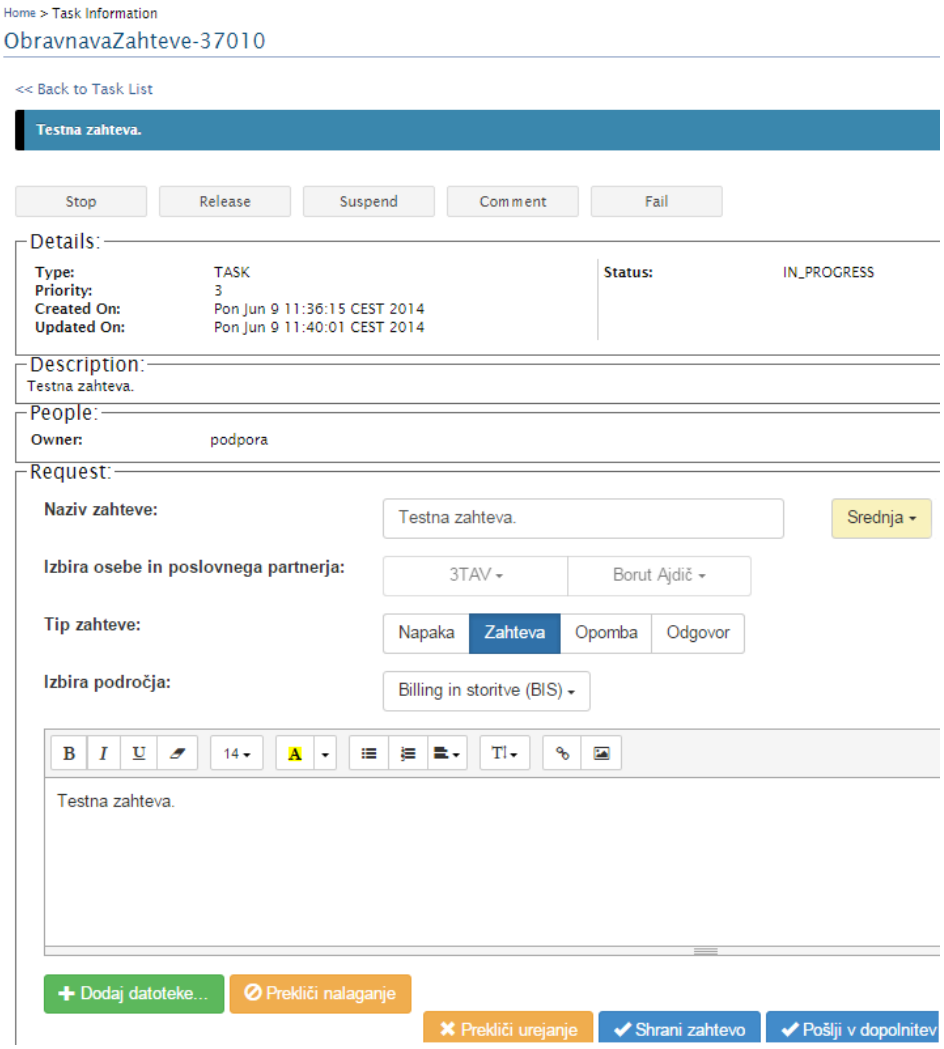
2. **Zaključevanje opravil**

   Po končani obdelavi podatkov posameznega opravila preko API z metodo *complete()* opravilo zaključimo.

6.4 Grafična predstavitev opravil

Človeška opravila omogočajo uporabo elementov za grafično predstavitev na dva načina. V praktičnem primeru smo uporabili oba. En način uporabe, ki ga sistemsko omogoča strežniška infrastruktura WSO2, je grafičen prikaz opravila na spletnem vmesniku za pregled opravil. Drugi način je bolj splošen in omogoča definiranje strukturiranih elementov za prikaz, ki jih s pomočjo dveh metod zunanji sistem pridobi preko programskega vmesnika za podporo odjemalcem.

1. Vgrajeni upravitelj opravil sistema WSO2 podpira splošni pregled opravil v obliki seznama. Uporabnik s svojo prijavo pridobi seznam opravil oziroma obvestil, nad katerimi ima pravico pregleda in izvajanja. Primer grafičnega prikaza vhodnih informacij zahteve iz našega primera je na sliki 6.3.



Slika 6.3: Primer grafičnega prikaza vhodnega sporočila opravila v sistemu WSO2

Informacije opravila prikažemo s pomočjo datotek JSP, ki podpirajo dinamično vsebino v jeziku HTML. Datoteke na strežnik namestimo skupaj z definicijo opravila. Podprte so tri različne datoteke:

- Datoteka **<ImeOpravila>-input.jsp** je namenjena izrisu vhodnega sporočila opravila. Elemente vhodnega sporočila pridobimo iz atributa opravila s funkcijo *request.getAttribute("taskInput")*

v obliki *OMElement* v programskem jeziku Java.

- Datoteka **<ImeOpravila>-output.jsp** definira uporabnikovo vnosno masko. Z implementacijo JavaScript funkcije *createTaskOutput = function() {};* pripravimo sistemu izhodno sporočilo v obliki XML. Funkcijo uporabi sistem, ko uporabnik potrdi vnesene podatke.
- Datoteka **<ImeOpravila>-response.jsp** je namenjena prikazu rezultata opravila oziroma grafični predstavitvi izhodnega sporočila opravila. Elemente izhodnega sporočila pridobimo iz atributa opravila s funkcijo *request.getAttribute("taskOutput").*
- Datoteko **<ImeOpravila>-Reminder-input.jsp** uporabimo ob obvestilu. Ker pri obvestilu samo prikazujemo informacije, za implementacijo drugih datotek ni na voljo.

2. Človeško opravilo omogoča tudi bolj splošno definiranje struktur za grafično predstavitev informacij opravila. Z elementom *htd:renderings* lahko v opravilo vključimo več svojih poljubnih tipov struktur, ki opisujejo grafični vmesnik. Struktura je torej definirana s tipom in predpisano shemo XSD tipa *xsd:any.* V našem primeru je struktura grafičnega vmesnika opisana v jeziku HTML.
   Zunanji odjemalci pridobijo grafične strukture preko programskega vmesnika za dostop do opravil. Metoda *getRendering(task identifier, rendering type)* glede na izbrano opravilo in želeni tip strukture vrne shemo strukture. Metoda je ista za opravila in obvestila. Seznam definiranih tipov struktur v obliki imenskih prostorov pridobijo z metodo *getRenderingTypes(task identifier).*
   Človeško opravilo tudi podpira, da se v strukturi uporabijo sistemske metode za dostop do lastnosti opravila, katerih rezultat postane del strukture in tako nudi zunanjemu klientu vse potrebne informacije za izris opravila.

3. V našem praktičnem primeru 6.11 smo na obvestilu definirali elektron-

ska sporočila, ki se pošljejo istočasno ob kreiranju obvestil. Na podlagi dogovorjenega imenskega prostora upravitelj opravil prepozna tip **wso2:email**, definiran v atributu elementa <*htd:rendering*>, in vrednosti uporabi za kreiranje elektronskega sporočila.

Koda 6.11: Definicija elektronskega sporočila v okviru obvestila

```
<htd:humanInteractions xmlns:wso2="http://wso2.org/ht/schema/renderings/">
  ...
  <htd:notifications>
    <htd:notification name="ObvestiloOZahteviNotification">
      <htd:renderings> <!-- podatki za elektronsko sporočilo -->
        <htd:rendering type="wso2:email"
        xmlns:wso2="http://wso2.org/ht/schema/renderings/">
          <wso2:to name="to" type="xsd:string">{$Prejemnik}</wso2:to>
          <wso2:subject name="subject"
          type="xsd:string">{$NazivZahteve}</wso2:subject>
          <wso2:body name="body"
          type="xsd:string">{$VsebinaObvestila}</wso2:body>
        </htd:rendering>
      </htd:renderings>
    </htd:notification>
  </htd:notifications>
</htd:humanInteractions>
```

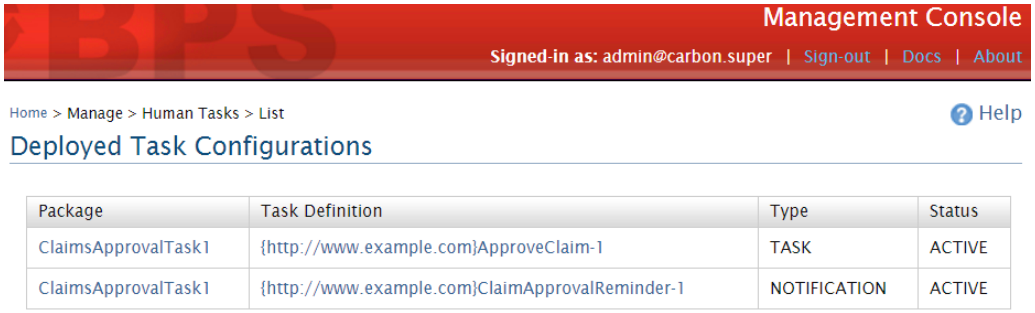
6.5 Izvajanje in spremljanje

Strežnik WSO2 omogoča s spletnim brskalnikom dostop do administrativnega modula. S tem modulom lahko nameščamo in konfiguriramo nove pakete procesov oziroma človeških opravil. Lahko nastavljamo lastnosti strežnika in spremljamo izvajanje nameščenih procesov.

Najprej smo definicijo procesa oziroma opravila skupaj s pomožnimi datotekami namestili na strežnik WSO2 (sliki 6.4 in 6.5). Pri namestitvi smo preverili sintaktično pravilnost kode in ustreznost konfiguracijskih datotek. Če je paket uspešno nameščen, strežnik glede na podano konfiguracijo od-

pre ustrezne porte in izpostavi povezavo do spletnih storitev za dostop do procesa oziroma opravil.

Slika 6.4: Namestitev paketa BPEL na strežnik WSO2



Slika 6.5: Seznam aktiviranih človeških opravil, nameščenih na strežniku WSO2

6.5.1 Avtorizacija procesa BPEL

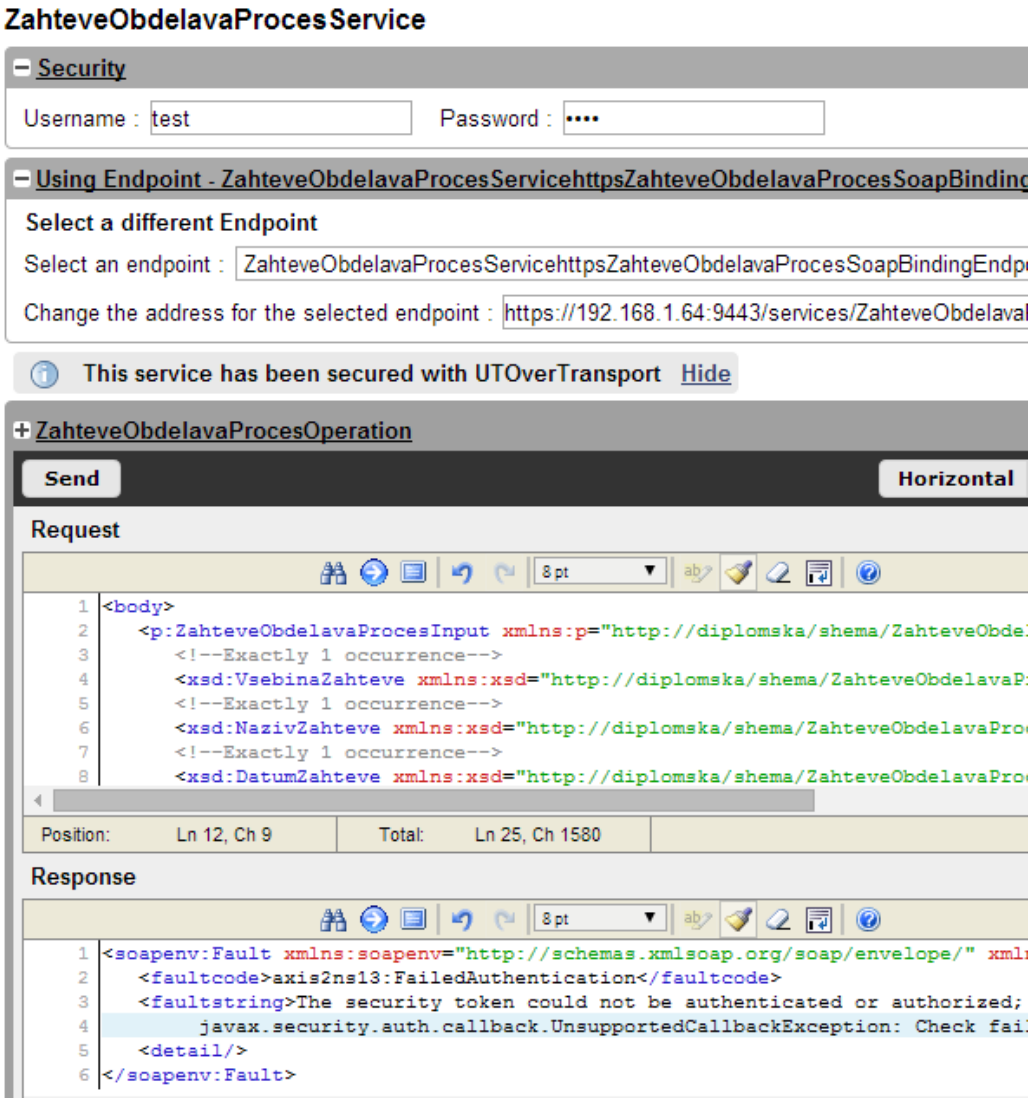
Ko smo na strežnik uspešno namestili proces BPEL, nam je poleg vseh informacij o nameščenem procesu na voljo tudi dodatna konfiguracija procesa.

Za avtorizacijo procesa BPEL imamo v okolju WSO2 na voljo več različnih varnostnih scenarijev (slika 6.6). V našem primeru smo uporabili avtorizacijo po principu uporabniškega žetona. Uporabnik se mora ob klicu procesa avtorizirati z uporabniškim imenom in geslom.

Slika 6.6: Nastavitev avtentikacije na procesu v sistemu WSO2

6.5.2 Oddaja zahteve s testnim orodjem

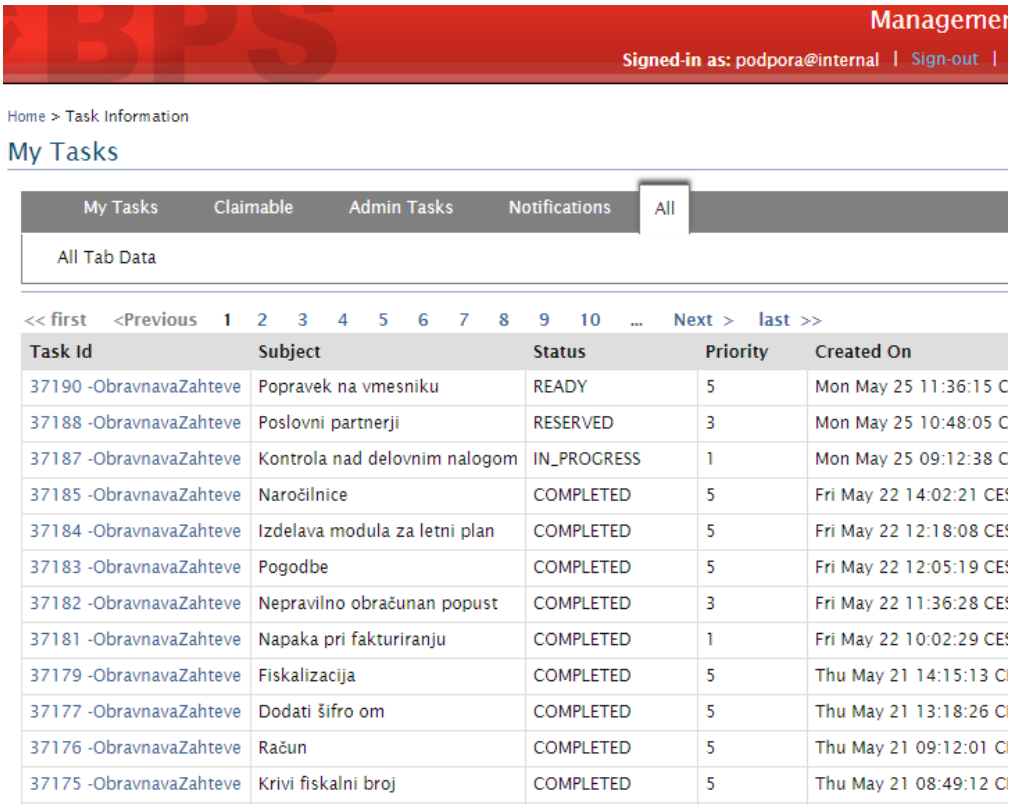
Nameščene pakete procesov BPEL oziroma opravil lahko v okolju WSO2 testiramo z vgrajenim testnim orodjem preko administratorskega modula. Glede na izbrani naslov spletne storitve so nam na voljo vse njene metode in njihova vhodna sporočila v formatu SOAP. Po oddanem sporočilu SOAP nam vmesnik izpiše tudi povratno sporočilo, ki je rezultat klica metode. Primer uporabe procesa BPEL v testnem okolju WSO2 je prikazan na sliki 6.7.



Slika 6.7: Primer oddaje zahteve in orodja za testiranje spletnih storitev v sistemu WSO2

6.5.3 Obravnava sprejetih zahtev

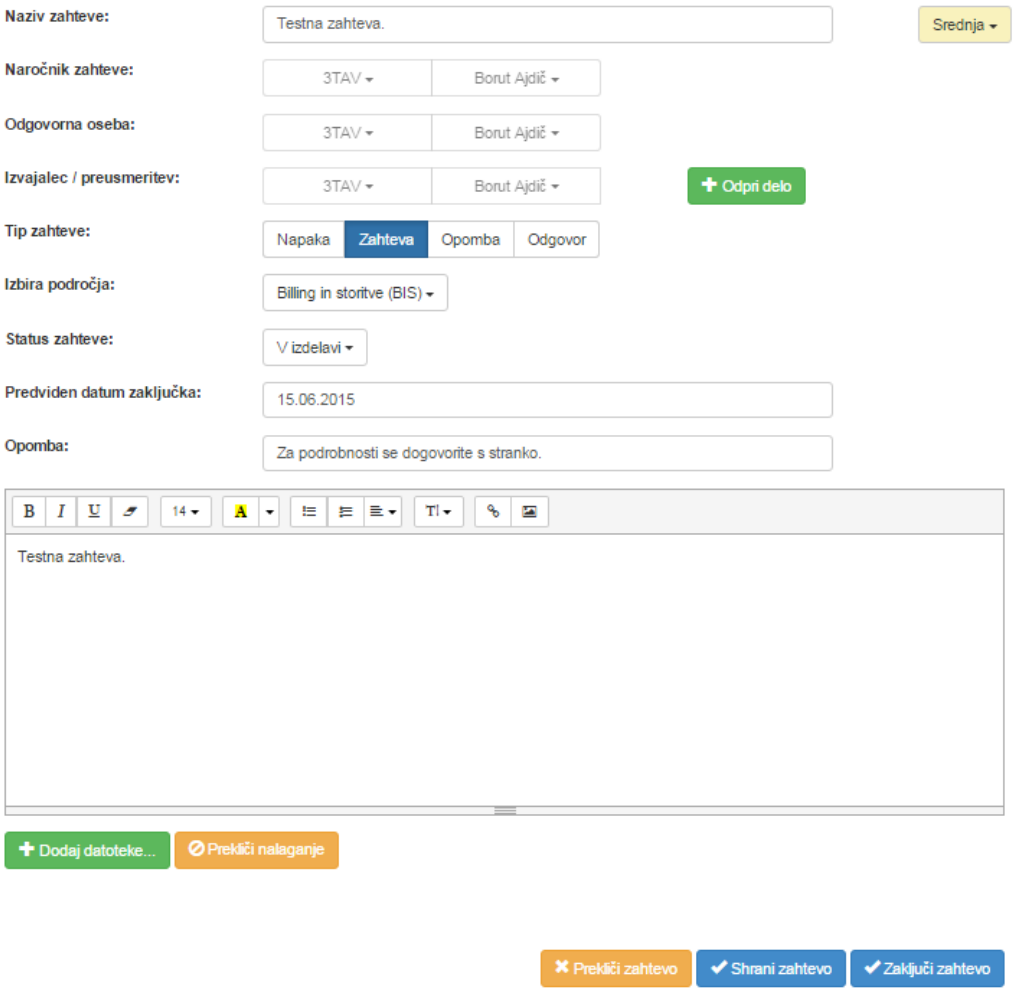
Zahteve strank so v obliki seznama na voljo referentom. Naloga referenta je obravnava posamezne zahteve in dopolnitev zahteve z obveznimi podatki. Pregled zahtev strank v sistemu WSO2 je prikazan na sliki 6.8.



Slika 6.8: Pregled sprejetih zahtev v sistemu WSO2

6.5.4 Reševanje zahteve

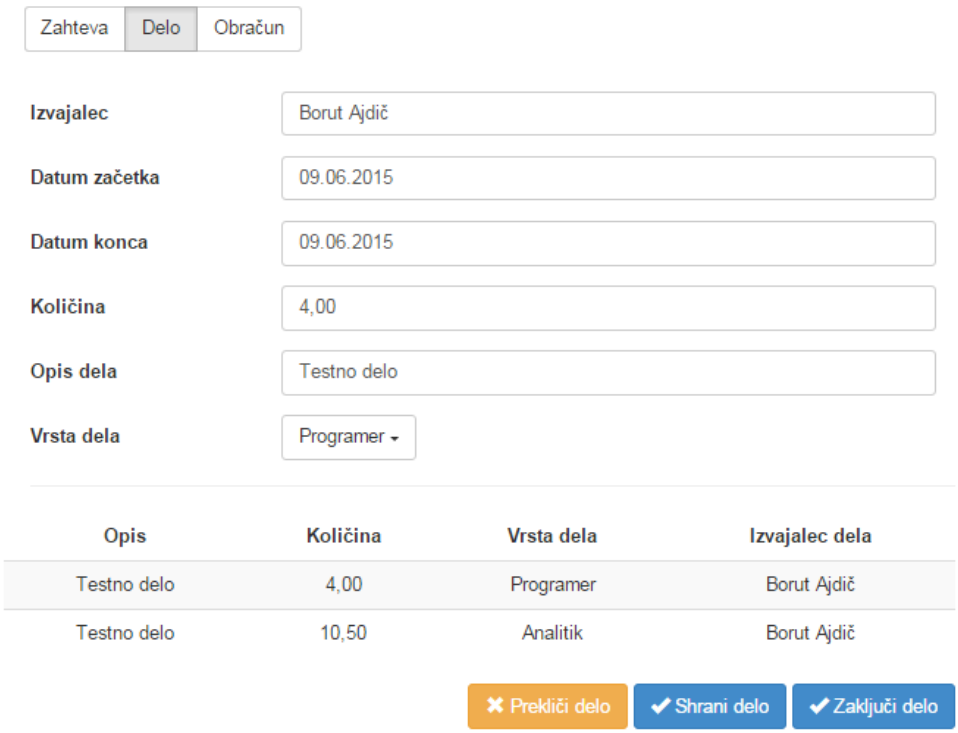
Na vmesniku za reševanje zahteve lahko odgovorna oseba ureja podatke o zahtevi, prilaga priponke na zahtevo in kreira delo. S tem se kreira podopravilo izbranemu izvajalcu. Zahtevo se lahko zaključi, ko so zaključena vsa dela na zahtevi. Zahtevo je mogoče tudi preklicati. Primer vmesnika za reševanje zahteve je prikazan na sliki 6.9.



Slika 6.9: Uporabniški vmesnik za reševanje zahteve

6.5.5 Urejanje podatkov o delu na zahtevi

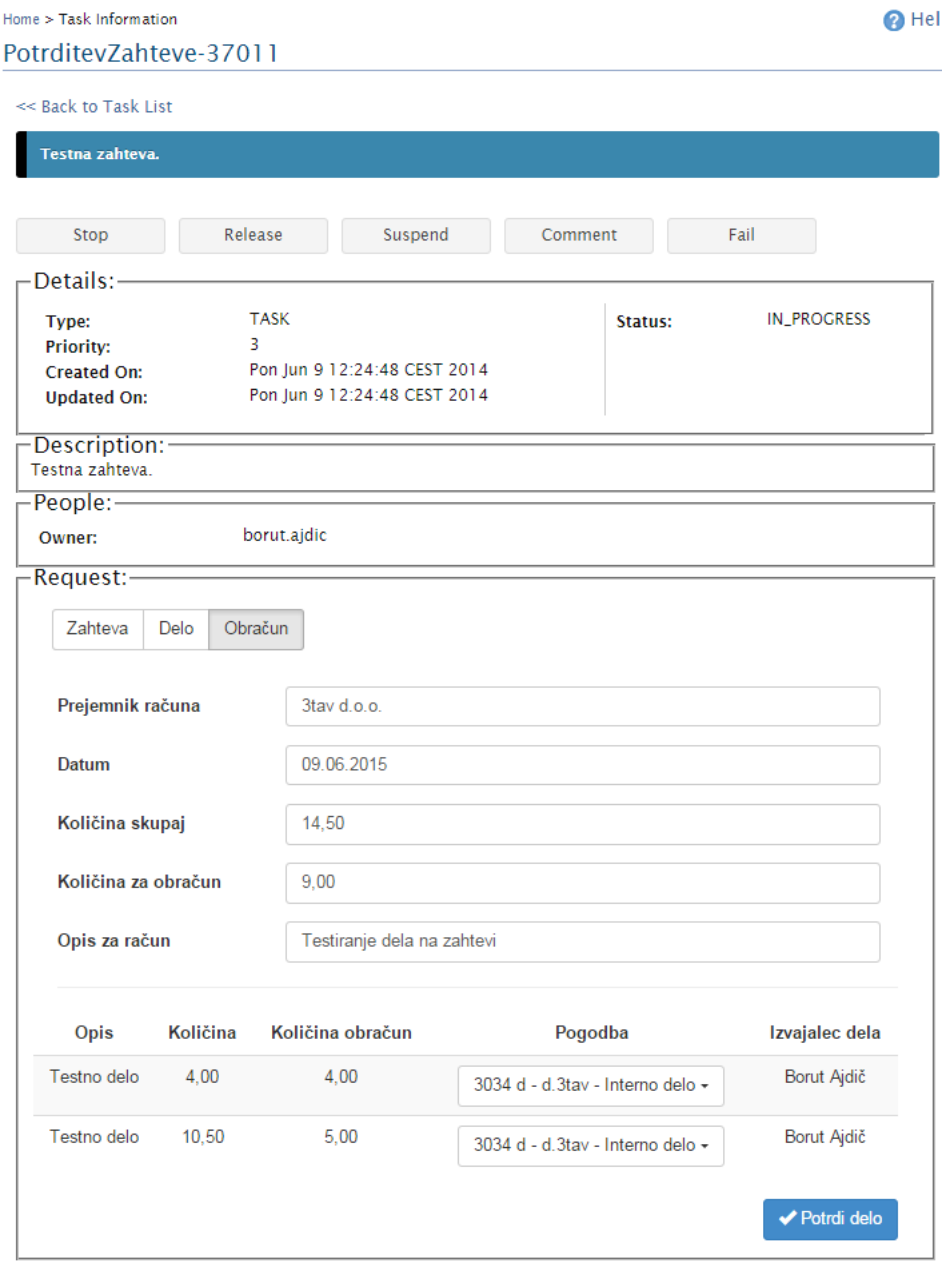
Primer vmesnika za urejanje podatkov o delu na zahtevi je prikazan na sliki 6.10. Referentu, ki naj bi opravljal delo na zahtevi, se kreira podopravilo za urejanje podatkov o delu na zahtevi. Referent ima možnost podatke o delu urejati in shranjevati, dokler dela ne zaključi ali prekliče. S tem se zaključi tudi podopravilo nadrejenega opravila za reševanje zahteve.



Slika 6.10: Uporabniški vmesnik za vnos podatkov o opravljenem delu na zahtevi

6.5.6 Potrditev dela na zahtevi

Zaključeno zahtevo morajo pred obračunom potrditi vsi deležniki na zahtevi, ki so opravili delo. Na zahtevi referent izbere ustrezno pogodbo za stranko in vpiše dejansko količino ur za obračun. Pri obračunu ur za stranko so upoštevane samo obračunske ure. Poslovni proces se nadaljuje šele, ko so potrjena vsa dela na zahtevi. Primer vmesnika za potrditev dela na zahtevi je prikazan na sliki 6.11.



Slika 6.11: Potrditev dela na zahtevi v sistemu WSO2

6.6 Pomanjkljivosti specifikacije WS-HumanTask

Med načrtovanjem in izvajanjem poslovnega procesa smo se srečali tudi s primeri uporabe, ki v osnovi niso podprti v specifikaciji WS-HumanTask. Ugotovljene pomanjkljivosti in njihove rešitve smo podrobneje predstavili v naslednjem seznamu:

1. Ni mehanizma, ki bi omogočal v procesu BPEL definiranje dodatnih uporabnikovih atributov. Te informacije bi opisovale zmožnosti uporabnika, na podlagi katerih bi dodeljevali lastništva nad opravili. V našem primeru smo to rešili z uporabo več človeških vlog. Želeno izbiro med njima smo opisali v parametru vhodnega sporočila opravila s poizvedbo XPath in jo upoštevali v pogojnem stavku pri postopku delegacije.

2. Množica potencialnih lastnikov omogoča, da za lastništvo nad opravilom tekmuje skupina uporabnikov. Če je potencialni lastnik samo eden, se predvideva, da je hkrati tudi že dejanski lastnik opravila. V tem primeru opravila ni mogoče samo ponuditi individualnemu resursu, kar se je v našem praktičnem primeru izkazalo za slabost. V splošnem sta uporabna oba principa, zato bi morala biti možnost izbire podprta v specifikaciji.

3. Delegiranje lastništva nad opravilom po sistemu naključne izbire iz množice uporabnikov ni podprto. Ena od možnih rešitev je uporaba lastnih funkcij v razširitvi jezika XPath 2.0, vendar specifikacija za zdaj predvideva samo uporabo jezika XPath 1.0.

4. Dodeljevanje opravil uporabnikom z najmanjšim številom opravil v čakalni vrsti ni podprto. S stališča optimizacije poslovnega procesa bi s tem mehanizmom dosegli enakomerno obremenjenost človeških resursov znotraj uporabniške skupine. Ena od možnih rešitev je seštevanje

opravil vsakega od potencialnih lastnikov in izbira tistega, ki ima dodeljenih najmanj opravil.

5. Mehanizem avtomatskega verižnega izvajanja ni podprt. Upravitelj opravil bi moral znati samodejno začeti naslednje opravilo v verigi opravil, potem ko se je prejšnje zaključilo. V našem praktičnem primeru smo to rešili z metodami *Complete* na prejšnjem opravilu in *Claim* na novem opravilu.

6. Trenutni lastnik opravila ne more zahtevati dodatnih človeških resursov na opravilu. Trenutno specifikacija predvideva simultano izvajanje več opravil, vendar lahko opravilo hkrati izvaja samo en uporabnik.

Poglavje 7

Zaključek

Razširitev izvršljivega jezika BPEL za modeliranje poslovnih procesov z uporabniško aktivnostjo je eden pomembnejših dosežkov na področju izvršljivih poslovnih procesov. Integracija človeka v izvedljivi poslovni proces na način, ki ohranja prvotno arhitekturo SOA in omogoča fleksibilnost ter robustnost, je zagotovo dodana vrednost pri razvoju distribuiranih sistemov. Človeška opravila kot samostojne komponente omogočajo v obliki spletnih storitev hitro in enostavno integracijo v različne sisteme.

Ugotovili smo, da je načrtovanje poslovnih procesov v jeziku BPEL različno od programiranja klasičnih poslovnih aplikacij in zahteva od programerja drugačna znanja. Velika prednost je zagotovo v neposredni preslikavi poslovnega procesa v izvedljivi jezik, saj lahko s pomočjo grafičnih komponent jezika BPEL našo rešitev orišemo tudi stranki in tako pospešimo fazi zajema funkcionalnih zahtev in načrtovanja.

Sistemi za izvajanje človeških opravil se med seboj razlikujejo po načinu implementacije mehanizmov za upravljanje s človeškimi opravili. Eni sistemi v celoti podpirajo specifikaciji BPEL4People in WS-HumanTask, spet drugi temeljijo na lastni interpretaciji. Negativne posledice razlik med ponudniki so v neprenosljivosti obstoječih procesov oziroma človeških opravil med sistemi in nekompatibilnost pri povezljivosti različnih sistemov. Razlike med različnimi ponudniki rešitev so s stališča razvijalca problematične

in nezaželene.

Sistem, na katerem smo preverili uporabnost človeških opravil v praksi, je eden od množice sistemov na trgu. Prednost sistema je v odprtokodni rešitvi, ki je zelo robustna, in jo je mogoče razširiti oziroma dopolniti. Sistem lahko enostavno namestimo na strežnik in s konfiguracijami prilagodimo svojim potrebam. Na voljo je zadovoljiv spletni vmesnik za administracijo in nadzor nad procesi in človeškimi opravili. Slabše se je izkazalo razvojno orodje, ki ne podpira grafičnega načrtovanja človeških opravil in je tudi zelo nestabilno.

Največji izziv pri načrtovanju in implementaciji je predstavljalo testiranje delovanja človeških opravil. Sistem v času pisanja diplomske naloge ni omogočal mehanizma, s katerim bi lahko simulirali opravila. Sintaktična pravilnost se preverja na strežniku pri sami namestitvi opravila in dostavi dovolj jasna sporočila o napakah. Ostale napake, ki se pojavijo med izvajanjem opravil, pa je zelo težko najti. Posledično je bilo testiranje zelo zamudno, saj je bilo treba ob vsakem popravku izvajati celoten poslovni proces.

V diplomski nalogi smo podrobneje opisali potrebo po integraciji ljudi v izvedljive poslovne procese in raziskali obstoječe tehnološke rešitve. Izbrani praktični primer zajema veliko konceptov, ki jih predvidevata specifikaciji BPEL4People in WS-HumanTask in jih tudi uspešno implementira. S tem smo pokazali nujnost mehanizma človeških opravil v izvedljivih procesih. Pri tem smo odkrili tudi določene pomanjkljivosti, ki bodo zagotovo v želji po standardizaciji in vse večji potrebi po podpori človeškim interakcijam v izvedljivih poslovnih procesih odpravljene v novih revizijah obstoječe specifikacije.

Slike

Tabele

Literatura

[1] OASIS. Web Services Human Task (WS-HumanTask), Specification Version 1.1. Dostopno na: http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cs-01.pdf, August 2010.

[2] Apache ODE. Dostopno na: http://ode.apache.org/, September 2015.

[3] ActiveVOS. Dostopno na: https://www.activevos.com/, September 2015.

[4] OASIS. WS-BPEL Extension for People (BPEL4People). Dostopno na: http://docs.oasis-open.org/bpel4people/bpel4people-1.1-spec-cs-01.pdfl, Avgust 2010.

[5] Gartner. Business Process Management (BPM). Dostopno na: http://www.gartner.com/it-glossary/business-process-management-bpm, 2013.

[6] OASIS. Koordinacijski protokol pri človeških opravilih. Dostopno na: http://docs.oasis-open.org/ns/bpel4people/ws-humantask/protocol/200803, Avgust 2010.

[7] W3C. Web Services Architecture. Dostopno na: http://www.w3.org/TR/ws-arch/, Februar 2004.

[8] M. B. Juric. BPEL: Service composition for SOA. Dostopno na: http://www.javaworld.com/article/2071789/soa/bpel--service-composition-for-soa.html, Julij 2006.

[9] L. Barroca, J. Hall, P. Hall, An Introduction and History of Software Architectures, Components, and Reuse, London, 2000.

[10] B. Brown, A. Garratt, B. Käckenmeister, M. Keen, A. Khairy, K. O'Mahony, L. Yu, Building IBM Business Process Management Solutions Using WebSphere V7 and Business Space, maj 2010.

[11] M. Gerhards, P. Pfeiffer, S. Skorupa, V. Sander. Towards a Security Framework for a WS-HumanTask Processor. Dostopno na: http://www.simpleweb.org/ifip/Conferences/CNSM/2011/papers/85707_1.pdf, 2011.

[12] S. Hrovat, Razvoj generične storitve za podporo sestavljenih uporabniških opravil v poslovnih procesih, Faktulteta za računalništvo in informatiko, Ljubljana, 2009.

[13] M. B. Juric, K. Pant, Business Process Driven SOA using BPMN and BPEL, Birmingham, avgust 2008.

[14] J. Listar, Razvoj informacijskih rešitev z uporabo BPM orodja IBM Websphere Lombardi Edition, Faktulteta za računalništvo in informatiko, Ljubljana, 2011.

[15] J. Mendling, K. Ploesser, M. Strembeck. Specifying Separation of Duty Constraints in BPEL4People Processes. Dostopno na: http://www.mendling.com/publications/08-BIS-paper_35.pdf, 2008.

[16] T. Sečnik, Načrtovanje in izvedba podpore poslovnim procesmo z orodji BPM, Faktulteta za računalništvo in informatiko, Ljubljana, 2012.

[17] jBPM. Dostopno na: http://www.jbpm.org, 2015.

[18] Intalio. Dostopno na: `http://www.intalio.com`, 2015.

[19] Oracle. Oracle SOA Suite. Dostopno na: `http://www.oracle.com/us/products/middleware/soa/suite/overview/index.html`, 2015.

[20] S. Perera. Human Tasks: Bridging Bits to Real World. Dostopno na: `http://wso2.com/library/articles/2012/01/human-tasks-bridging-bits-real-world/`, Januar 2012.

[21] OASIS. WS-Trust 1.3. Dostopno na: `http://docs.oasis-open.org/ws-sx/ws-trust/200512`, Marec 2007.

[22] OASIS. WS-BPEL for People (BPEL4People). Dostopno na: `http://xml.coverpages.org/bpel4people.html`, 2009.

[23] XSD Date and Time Data Types. Dostopno na: `http://www.w3schools.com/schema/schema_dtypes_date.asp`, 2015.

[24] OW2 Orchestra. Dostopno na: `http://orchestra.ow2.org`, 2015.

[25] Vmesnik API za podporo odjemalcem v sistemu WSO2. Dostopno na: `http://localhost:9443/services/HumanTaskClientAPIAdmin?wsdl`, 2015.

[26] World Wide Web Consortium, Service Oriented Architecture, Web Services Architecture Group, 2003.

[27] W3C. Extensible Markup Language (XML) 1.0 (Fifth Edition). Dostopno na: `http://www.w3.org/TR/REC-xml/`, November 2008.

[28] W3C. Web Services Architecture Requirements. Dostopno na: `http://www.w3.org/TR/wsa-reqs/`, Februar 2004.

[29] T. Baeyens, M. V. Faura. The Process Virtual Machine. Dostopno na: `http://www.onjava.com/pub/a/onjava/2007/05/07/the-process-virtual-machine.html`, November 2007.

[30] OASIS. V. Faura. UDDI. Dostopno na: http://uddi.xml.org/, 2015.

[31] W3C. XML Schema Part 1: Structures Second Edition. Dostopno na: http://www.w3.org/TR/xmlschema-1/, 2004.

[32] W3C. XML Schema Part 2: Datatypes Second Edition. Dostopno na: http://www.w3.org/TR/xmlschema-2/, Oktober 2004.

[33] WSO2. Dostopno na: http://wso2.com, 2015.

[34] Architecture diagram for WS-HumanTask engine. Dostopno na: http://wso2-oxygen-tank.10903.n7.nabble.com/WSO2-BPS-WS-HumanTask-support-Architecture-review-td30200.html, Januar 2012